(19)日本国特許庁（ＪＰ）　(12) 公開特許公報（A）　(11)特許出願公開番号

特開平10－49416

(43)公開日　平成10年(1998) 2月20日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　12/00	５３１		Ｇ０６Ｆ　12/00	５３１Ｍ	

審査請求　未請求　請求項の数23　ＯＬ　外国語出願　（全135頁）

(21)出願番号　特願平8－316853

(22)出願日　平成８年(1996)10月23日

(31)優先権主張番号　５４６７２７　5,719,395

(32)優先日　1995年10月23日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　596144584
スタック　エレクトロニクス
Ｓｔａｃ　Ｅｌｅｃｔｒｏｎｉｃｓ
アメリカ合衆国　カリフォルニア州
92130－2093　サンディエゴ　スイート
400　ハイ　プラフ　ドライヴ　12363

(72)発明者　ダグラス　エル　ホワイティング
アメリカ合衆国　カリフォルニア州
92009　カールスバッド　フェボ　コート
3312

(74)代理人　弁理士　柳田　征史　（外１名）

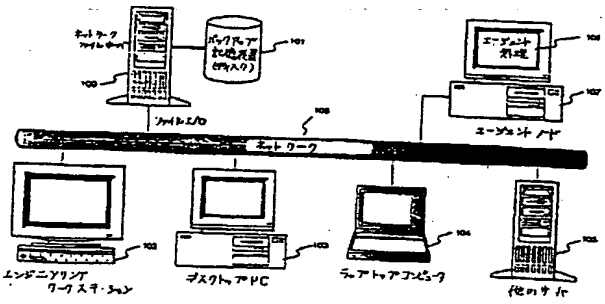
最終頁に続く

(54)【発明の名称】　ネットワークシステムにおけるファイルの　バックアップ方法

(57)【要約】　（修正有）

【課題】　コンピュータネットワークの多数のノード上のディスクボリュームからのファイルを共通ランダムアクセスバックアップ記憶手段にバックアップする。

【解決手段】　特定のボリュームの初期バックアップ後の各バックアップ動作に関しては、前のバックアップ以降変更されたファイルのみが実際にボリュームから読み出され、バックアップ記憶手段に記憶される。これらのエンハンスメントの全ては、バックアップを実行するのに必要とされる記憶容量及びネットワーク帯域幅量の両方を著しく減少する。バックアップからのファイルを見るか又は復旧するために、ユーザは、バックアップの時点で元のディスクボリューム全部のディレクトリ構造と同一のディレクトリ構造を有するディスクボリュームとしてバックアップセットを取り付けることができる。



1

【特許請求の範囲】

【請求項１】　バックアップ記憶手段のためにコンピュータネットワークのノードのディスクボリュームに記憶されたデータファイルをバックアップする方法であって、前記バックアップ記憶手段が、前記コンピュータネットワーク上の他のノードから既にバックアップされたデータファイルを含むものにおいて、前記方法が、

前記ディスクボリュームからバックアップされるファイルと照合するために前記バックアップ記憶手段に既に含まれている前記ファイルのリストを端から端まで探索するステップと、

前記ディスクボリュームからバックアップされるファイルと前記リストの中に既に含まれた前記ファイルのいずれかとの間でいかなる一致も得られない場合に作動して、前記バックアップ記憶手段にバックアップされる前記ファイルの内容の全表示を記憶し、前記全表示の前記バックアップ記憶手段上の位置を示すインデックスを計算し、かつ前記ディスクボリュームからバックアップされる前記ファイルを記述するエントリを前記リストに付加するステップと、

前記ディスクボリュームからバックアップされるファイルと前記リストの中に既に含まれた前記ファイルとの間で一致が得られる場合に作動して、前記リストの中に既に含まれた前記ファイルの内容の全表示の前記バックアップ記憶手段上の位置を示すインデックスを計算するステップと、

バックアップ動作の時間に前記ディスクボリュームのディレクトリ構造を指定するデータ構造を記憶するステップとからなり、前記データ構造が、前記ディスクボリュームからバックアップされた各前記ファイルに対して、前記全表示の位置を示す前記インデックスも含み、バックアップされた前記ファイル又は前記リストの中に既に含まれた前記ファイルのどちらかは前記リストの端から端までの前記探索の結果に依存し、かつそれによって、ノードにわたって複写されたファイルが、前記ファイルの内容の一つのコピーだけが前記バックアップ記憶手段に記憶されるように識別することができることを特徴とする方法。

【請求項２】　バックアップされる前記ファイルの内容の前記全表示を記憶するステップが、

前記ファイルの前のバージョンが前記ノードから前記バックアップ記憶手段に既にバックアップされた場合に作動して、前記ファイルの前のバージョンとの差を計算し、インデックスを使用してバックアップされる前記ファイルの内容の一部を表示して、前記バックアップ記憶手段の前記前のバージョンの表示とするステップをさらに含むことを特徴とする請求項１記載の方法。

【請求項３】　前記ファイルの前記前のバージョンの存在が、前のバックアップ動作のディレクトリ構造を指定する予め保存されたデータ構造を使用して検出され、かつ前記バージョン間の差が、前記ファイルの前記前のバージョンの内容の全表示を指し示す前記予め保存されたデータ構造に含まれた、インデックスを使用して計算されることを特徴とする請求項２記載の方法。

2

【請求項４】　バックアップされる前記ファイルの内容の前記全表示を記憶するステップが、前記バックアップ記憶手段に前記表示を記憶する前に無損失データ圧縮アルゴリズムを使用して前記表示の部分を圧縮するステップをさらに含むことを特徴とする請求項３記載の方法。

【請求項５】　バックアップされる前記ファイルの内容の前記全表示を記憶するステップが、バックアップされる前記ファイルの内容に関するハッシュ関数を計算することによって得られるハッシュ暗号化キーを使用して前記全表示の部分を暗号化するステップをさらに含むことを特徴とする請求項４記載の方法。

【請求項６】　前記ハッシュ暗号化キーが、前記ディスクボリュームのディレクトリ構造を指定する前記データ構造の一部として記憶されていることを特徴とする請求項５記載の方法。

【請求項７】　バックアップされる前記ファイルの内容の前記全表示を記憶するステップが、バックアップされる前記ファイルの内容に関するハッシュ関数を計算することから得られるハッシュ暗号化キーを使用して前記全表示の部分を暗号化するステップをさらに含むことを特徴とする請求項１記載の方法。

【請求項８】　前記ハッシュ暗号化キーが、前記ディスクボリュームのディレクトリ構造を指定する前記データ構造の一部として記憶されていることを特徴とする請求項７記載の方法。

【請求項９】　前記ディスクボリュームの前記構造を指定する前記データ構造を記憶するステップが、

無損失データ圧縮アルゴリズムによって前記データ構造の部分を圧縮するステップと、

前記コンピュータネットワーク上の前記ノードに専用な暗号化キーを使用して前記ハッシュ暗号化キーを暗号化するステップとをさらに含むことを特徴とする請求項１～８のいずれか記載の方法。

【請求項１０】前記バックアップ記憶手段の中に既に含まれている前記ファイルの前記リストが探索時間を最少にするためにデータベースとして構成されていることを特徴とする請求項１～８のいずれか記載の方法。

【請求項１１】前記データベースにおける各エントリが、ファイル名、長さ及び作成の時間を含む前記エントリに関連した前記ファイルのためのディレクトリエントリ情報で計算されたハッシュ関数と、前記ファイルの内容の部分に関して計算されたハッシュ関数とを含むことを特徴とする請求項１０記載の方法。

【請求項１２】前記データベースの前記探索が、下記のステップ、すなわち、　前記データベースの第１のセクションをロードするステップであって、前記第１のセク

ションが部分エントリを含み、各部分エントリが前記データベースのエントリの一部だけを含むことと、

バックアップされる前記ファイルの新データベースエントリを生成するステップと、

前記新データベースエントリと前記部分エントリとの間で一致をとるために前記第１のセクションの端から端まで探索し、かつ前記新データベースエントリと前記第１のセクションの部分エントリとの間の一致が得られた場合作動して、前記データベースの第２のセクションの中の関連エントリから前記一致部分エントリの残りの部分をロードし、かつ前記新データベースエントリと全データベースエントリとの間で全一致があるかどうかを決定するために前記新データベースエントリと前記関連エントリの前記残りの部分とを比較するステップとを含むことを特徴とする請求項１１記載の方法。

【請求項１３】前記データベースの前記第１のセクションが、前記部分エントリのビットフィールドに基づくソート順序で記憶され、かつ無損失データ圧縮アルゴリズムで圧縮されていることを特徴とする請求項１２記載の方法。

【請求項１４】前記無損失データ圧縮アルゴリズムが、前記ソートされた第１のセクションの中のいかに多数の連続エントリが、前記ビットフィールドの各可能値のビットフィールドを有しているかを示すアレイを記憶することを含み、かつ前記第１のセクションの残りものが前記部分エントリの残りから前記ビットフィールドを省くことを特徴とする請求項１３記載の方法。

【請求項１５】特定のバックアップ動作の内容が、前記バックアップ動作の時点で最初のディスクボリュームのディレクトリ構造と同一のディレクトリ構造を有する復旧ディスクボリュームとして備えられていて、それによって前記復旧ディスクボリュームの前記ファイルが、通常のファイルシステム入出力呼び出しを使用するいかなるアプリケーションソフトウェアからもアクセスすることができることを特徴とする請求項１～８のいずれか記載の方法。

【請求項１６】前記復旧ディスクボリュームが読み出し専用方式でアクセスできることを特徴とする請求項１５記載の方法。

【請求項１７】前記復旧ディスクボリュームが読み出し及び書き込みに対してアクセスでき、前記復旧ディスクボリュームに対する全ての書き込みが、前記復旧ディスクボリュームが取り付けられていない場合、その内容が廃棄される一時記憶手段にキャッシュされていることを特徴とする請求項１５記載の方法。

【請求項１８】前記バックアップされる前記ファイルと前記ファイルの前記前のバージョンとの差が確率アルゴリズムを使用して計算され、下記のステップ、すなわち、

前記前のバージョンがバックアップされる時点で、前記

バックアップ記憶手段に前記前のバージョンの一定サイズのチャンクに関して計算されたハッシュ値セットを記憶するステップと、

前記バックアップの時点で、前記予め記憶されたハッシュ関数結果をロードするステップと、

前記前のファイルバージョンからの前記ハッシュ関数結果とバックアップされる前記ファイルの一定サイズのチャンクに関して計算されたハッシュ関数計算とを比較するステップと、

バックアップされる前記ファイルのチャンクが前記前のファイルのチャンクとおなじハッシュ値を有する場合、作動して、前記前のバージョンの前記一致のチャンクを示すインデックスによってバックアップされる前記ファイルの前記チャンクを表示するステップとを含むことを特徴とする請求項２～８のいずれか記載の方法。

【請求項１９】前記ハッシュ関数結果の前記比較が、バックアップされる前記ファイル内のバイト毎にハッシュ関数計算をスライドさせることを含み、それによってバックアップされる前記ファイル内の一致チャンクがバックアップされる前記ファイル内のいかなるバイト境界上でも得ることができ、単独でチャンク境界上で得ることができないことを特徴とする請求項１８記載の方法。

【請求項２０】前記ハッシュ関数が周期冗長検査（CRC）の計算を含んでいることを特徴とする請求項１９記載の方法。

【請求項２１】特定のバックアップ動作の内容が、前記バックアップ動作の時点で最初のディスクボリュームのディレクトリ構造と同一のディレクトリ構造を有する復旧ディスクボリュームとして備えられていて、それによって前記復旧ディスクボリュームの前記ファイルが、通常のファイルシステム入出力呼び出しを使用するいかなるアプリケーションソフトウェアからもアクセスすることができることを特徴とする請求項１８記載の方法。

【請求項２２】前記復旧ディスクボリュームが読み出し専用方式でアクセスできることを特徴とする請求項２１記載の方法。

【請求項２３】前記復旧ディスクボリュームが読み出し及び書き込みに対してアクセスでき、その内容が廃棄される前記復旧ディスクボリュームに対する全ての書き込みが、前記復旧ディスクボリュームが取り外される場合、一時記憶手段にキャッシュされていることを特徴とする請求項２２記載の方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、コンピュータネットワーク上の複数のノードが共通ランダムアクセスバックアップ記憶手段にファイルをバックアップすることを可能にするシステムに関するものである。

【０００２】

【従来の技術】コンピュータディスクからのデータ及び

5

プログラムファイル（ここではしばしばともに“データ”と呼ばれる）をバックアップすることは、長年の間、周知のやり方であった。データがバックアップされることを必要とする２つの主要な理由がある。第１の理由は、ディスクハードウェアが故障して、ディスクに記憶された有用なデータのどれにもアクセスできなくなるかも知れないということである。この悲惨なタイプの事象は、しばしば悲惨的な故障と呼ばれる。この場合、バックアップが実行されたと仮定すると、一般的には、コンピュータ操作者は全てのファイルを最も最新のバックアップから“復旧”する。幸いにも、新しいコンピュータディスク及びコントローラは長い年月にわたってより信頼できるようになってきているが、このような災難の可能性はなお無視することができない。バックアップのための第２の理由は、ユーザが重要なデータファイルを誤って削除又は上書きするかも知れないということである。この種の問題は、通常、悲惨的なハードウェア故障よりも非常に一般的であり、一般的には、コンピュータ操作者はバックアップ媒体（例えば、テープ）から元のディスクへ破壊されたファイルだけを復旧する。

【０００３】一般に、フロッピーディスク駆動技術及び他の交換可能ディスク駆動技術（例えば、ベルノウリー、シクエスト、光学的）も使用されるけれども、バックアップ装置はテープ駆動装置である。テープは、（媒体だけのコストを考慮し、駆動装置のコストを無視する場合）記憶装置のバイト当たりより低いコストを有する利点を有し、その理由のために、テープは、大部分のアプリケーション、特に、ネットワークファイルサーバのような大量のデータが必要とされるアプリケーションでは好ましい。テープは主に逐次アクセス媒体である。すなわち、もし出来るとすれば、ランダムアクセスは、通常、ディスク駆動装置が数ミリ秒であるのとは対照的に約数10秒のオーダー（分のオーダーではないが）の時間を必要とする。同様に、移動テープを停止及び再始動するための時間は約数秒のオーダーであるので、許容し得るバックアップ性能を確実にするためにテープ駆動を“流し”続けるのに十分なデータを供給することは重要なことである。バックアップの完了後に、テープカートリッジの安全を確保するためにオフサイトに持ち出される。所与のバックアップからデータを復旧するための要求が生じる場合、適当なテープカートリッジはテープ駆動装置に再挿入され、ユーザは、順にテープから検索され、かつディスクボリュームに書き込まれる復旧されるべきファイルを選択する。

【０００４】テープカートリッジのセットを安全環境で物理的に保管し、復旧するために必要とされるテープの選択を容易にするためにテープカートリッジの目録を作るための作業は、バックアップソフトウェア及びバックアップ管理者（すなわち、バックアップ処理及び方式を実施する責任を負うべき個人）の両者の重要な（そして

6

しばしば魅力的な）機能である。さらに、バックアップ動作又は復旧動作が複数テープを必要とするならば、バックアップ管理者によって手動的又はテープジュークボックス（すなわち、ロボットテープ自動交換機）を使用して自動的のいずれかでテープを交換できる機能を備えなければならない。したがって、テープの交換は、給料又はジュークボックスロボット工学のいずれかのために相当な直接経費並びに通常数十秒以上のかなりの時間遅れを伴い得ることである。

【０００５】バックアップ時間並びに使用されたテープ量を節約するために、いろいろな種類の“増分（incremental）”バックアップ方法を使用することができる。例えば、一般的な操作は、毎週１回ディスクボリューム上の全てのファイルの完全なバックアップを実行し、次にその週の次の日以降は最後のバックアップ以来変えられたファイルだけをバックアップすることを含んでいる。この考えに関する他の変形は、各部分バックアップが前の部分バックアップの代わりに最後の完全なバックアップ以降の全ての変更を含む“差分（differential）”バックアップとして知られている。すなわち、この方法は、特定のバックアップ時点のファイルに復旧するために２つのバックアップ（１つは完全及び１つは部分）をアクセスするだけで良い。大抵の場合、１日当たりディスクボリューム上で実際変わるデータ量は全体の中のほんの少しであるので、このような方式は、増分バックアップが実行された日に、バックアップ“ウィンドウ”と又はバックアップに必要な時間量とを著しく減少させるという利点がある。さらにまた、完全バックアップ及びいくつかの増分バックアップの全てからのデータを単一のテープカートリッジ上で適合させることができ、完全バックアップと次の完全バックアップの間の曜日にいかなるテープ交換に対する要求も不要にすることができる。ディスクボリューム及びテープ駆動装置がネットワークを通じて接続された別々のコンピュータにある場合、増分バックアップはネットワーク帯域幅要求もかなり減少する。

【０００６】増分バックアップが時間及び媒体を節約ことができることは本当であるが、増分バックアップはまた完全バックアップよりも使用するのがしばしば非常に困難である。ユーザの観点から、各増分バックアップに含まれるファイルセットは、通常、いかにユーザがユーザのディスクボリュームの内容を見るかには全く関係ない。換言すれば、所定のファイルは最後のバックアップ以来変更されるけれども、ディスクボリュームは、変更済ファイル及び未変更ファイルの全てのファイルの完全コピーをなお含んでいる。全てのファイルのいずれも所与の操作を実行するのに必要とされる。あいにく、従来技術の増分バックアップを処理する復旧ソフトウェアは、一般的には、ユーザに変更済ファイルだけの一覧を表示し、増分バックアップの時点でディスクにある全て

7

のファイルの混合一覧を表示しない。したがって、たとえば、全サブディレクトリに格納された当該ファイルを、当該増分バックアップの日付の状態に戻したければ、直前の完全バックアップと途中の増分バックアップを利用して、個々のファイルを最新状態に戻さなければならない。同様に、ユーザがバックアップテープからファイルのセットを識別したいならば、ユーザは、通常、関心のあるファイルを全てを探すためにいくつかの増分／完全バックアップセットを通読しなければならない。一旦ファイルが選択されると、たとえファイルがディスク及び完全バックアップ上で全て連続しているとしても、ファイルはテープ一面に非常に適当に分散することができ、したがって、非常に遅い復旧処理を生じる。これらの理由のために、増分バックアップは、しばしばせいぜい不承不承に使用され、たまにではなく、それは常に完全なバックアップを実行することのために拒否される。

【０００７】復旧を実行する際の他の重要な制限は、いかにユーザがテープ上に記憶されたファイルをアクセスできるかと関係がある。一般的には、復旧ファイルは、ユーザが自分のファイルを選択し、次にテープからディスクボリュームへファイルを復旧することを可能にするバックアップソフトウェアパッケージの一部として提供された特別のアプリケーションを実行することを必要とする。ユーザは復旧アプリケーションをたまに実行するために、復旧アプリケーションはファイルの処理に不慣れなインタフェースを提供し、他のアプリケーションプログラムで直接にファイルをアクセスすることができない。大部分のユーザは、ワードプロセッサ、ファイルマネージャ、スプレッドシート等を含むファイルを見て、処理するための自分自身のお気に入りのアプリケーションセットを既に持っているので、ユーザが自分自身のツールを使用してファイルを見て、選択し、復旧することを可能にするように擬似ディスクボリュームとしてバックアップ画像を“備える”という概念は、魅力的に思われ、２、３の事例で実施されている（例えば、コロンビアデータのスナップバック製品、参照することによってここに組み込まれている1995年10月４日出願され、本発明の譲受人に譲渡された名称が「コンピュータディスクボリュームをバックアップするためのシステム」である米国特許出願）。しかしながら、このようなランダムアクセスアプリケーションにおけるテープの固有遅延は、この概念の有用性を幾分限定されたものにする。増分バックアップが完全バックアップ内にあるよりもさらに広くファイルをテープ上に分散して記録するならば、このやっかいさは特に激化される。

【０００８】単一のスタンドアローンコンピュータの場合、バックアップのための構成は、バックアップ装置（例えば、テープ駆動装置）をコンピュータに付加することからなる。しかしながら、ネットワーク環境では、

8

この状況はなおさら複雑であり、多くの構成は、コスト、取り扱いやすさ及び帯域幅（テープ及びネットワークの両方とも）において複雑なトレードオフを処理しようする試みで使用されている。大部分のコンピュータネットワークは、ファイルサーバ又はアプリケーションサーバであるノード並びにユーザのワークステーション（例えば、デスクトップパーソナルコンピュータ）であるノードを含んでいる。通常、サーバは会社又は部門全体のための重要なデータを含んでいるので、サーバをバックアップすることは絶対に必要な作業であり、通常、ネットワーク管理者によって処理される。各サーバが、そのディスクをバックアップするための専用テープ駆動装置を有することは珍しいことでないが、多くの場合、単一テープ駆動装置は、ネットワークを通じてバックアップデータを送信することによって複数のサーバをバックアップするために使用することができる。前者の方式は、より高価で、多くの場所でのテープカートリッジを管理することを必要とするが、前者の方式は、ネットワークを通じて供給するデータを高速度テープ駆動装置に流し続けることをしばしば不可能にする後者の方式におけるネットワーク帯域幅制限を避ける。駆動装置コスト、媒体コスト、テープ駆動速度、ネットワーク帯域幅、バックアップの頻度、ハードディスクのサイズ、バックアップウィンドウの許容範囲等を含む必要とされるいろいろな要因の混合により、多くのシステムが技術が進歩するにつれて発展する方式の混ざり合ったもの利用することは意外なことではない。

【０００９】ネットワーク上のパーソナルコンピュータワークステーションも重要なデータもしばしば含んでいる。このようなデータ量は、ディスク駆動容量コストの連続する劇的な減少のために平均のワークステーションディスク容量が増大するにつれて増加する。今日の殆どのネットワーク上のワークステーションから送られるデータは、いくつかの実行可能なバックアップ手段があるにも拘らず、気まぐれに思いついた時だけバックアップされている。たとえば、ワークステーション毎にテープ装置を用意（接続）することで、一応の問題解決となるが、実際には、この方法でもたらされる効果を阻害する、数多くの障害が立ちはだかっている。これらの問題の中には、駆動装置コスト、媒体コスト、エンドユーザ訓練コスト及びバックアップテープを管理する困難さがある。このバックアップテープは、定期的にバックアップを行うというユーザ教育の欠如はさておき、組織全体に亙って、必要に応じ散らばっている。

【００１０】ほとんどあらゆるネットワークバックアップソフトウェアパッケージの一部として含まれる他の方法によって、ワークステーションデータはネットワークを通じて共有バックアップ装置にバックアップすることができ、したがって、駆動装置と媒体の集中管理を可能にし、多数のユーザ間のハードウェアコストを償却す

る。しかしながら、そのすぐに役立つ有用性にもかかわらず、この技術は、いろいろな理由のためにわずかな割合の設置だけに使用される。例えば、ネットワークが何ダースものワークステーション又は数百のワークステーションさえも含むことは珍しいことではない。その場合において、適度なウィンドウ（例えば、オーバーナイト（overnight））内でワークステーションの全てをバックアップするのに十分なネットワーク帯域幅がないこともある。さらに、一般的に必要とされる完全なデータ量は、ハードウェア及びテープ管理のコストを大いに増加するテープジュークボックスの使用を強制する。さらにまた、特に、一方ではテープ駆動を流し続けるのに十分高速でデータを供給する必要があるが、他方では、バックアップ動作は、一般的には、ユーザ応答及びネットワーク応答の速度を落とすので、各ワークステーションのためのテープ駆動装置の使用をスケジュールする際に競合することがある。米国特許第5,212,772号においてギガトレンド(Gigatrend)によって特許となった一つのシステムは、テープを流し続けることができることを確実にしようとする試みで単一のテープカートリッジ上に複数のワークステーションからのデータをインタリーブすることによってこの問題のほぼ一部を解決しているが、この方式は、ほとんど商業的成功をおさめていない。多分受け入れるための主要な障害は、一旦ユーザが自分のワークステーションの前のバックアップからデータを復旧する必要があると決心すると、もし非常に大きい（そして高価な）テープジュークボックスが使用されなくて、かつ適切なソフトウェアがジュークボックスを遠隔に操作するのに使用可能でないならば、ユーザは関心のあるテープを選択するための媒体の制御権を有していないという簡単な事実である。手動で、この作業においてユーザを助けることが、ネットワーク管理者の優先順位リスト上に非常に高くランク付けされることはまれであり、それは他のスケジュール済のテープ駆動装置の使用とも競合することがある。すなわち、この結果、ほとんど必ず、復旧されるデータを最終的にアクセスする際の遅延はエンドユーザ及びネットワーク管理者の両者を失望させたままであるということである。

【００１１】ディスク駆動容量のコストがますます減少するにつれ、ワークステーションバックアップ問題に対する他の解決策がいくつかのネットワークで最近とられている。ネットワーク管理者は、大きなディスク駆動装置(又は駆動装置セット)をネットワーク上のファイルサーバに付加し、ユーザは自分のワークステーションからのファイルを新しいディスクのサブディレクトリトリーに単にコピーする。所望ならば、バックアップデータの機密は、各ユーザのディレクトリに標準ネットワークセキュリティアクセス権を割り当てることによって確実にされる。サーバに配置されたファイルは、規則的なサーババックアップ処理の一部としてバックアップされ、必

要に応じて第２のレベルのデータ障害復旧を実現できる。各ユーザは、ネットワーク管理者によるいかなる介入もなしに、自分自身の好ましいアプリケーションを使用してネットワーク上の自分のバックアップディレクトリからのファイルを容易にアクセスできる。この一般的な方式は所望ならばサーババックアップにも適用できることに注目されたい。現在の価格では、ローエンドテープ駆動装置の価格に匹敵する価格で各ユーザは１ギガバイトのディスク領域を付加することができる。このメガバイト当たりコストに基づいた解決策は、ここで論議された他の解決策よりもより高価であることもあるが、それにもかかわらず、このコストはいくつかの環境で受け入れることができる。この方法は、ネットワーク帯域幅制約、全ての重要なファイルを規則的にバックアップする際のユーザの訓練に対する要求、及び一般的には管理者の助けを必要とするテープをアクセスしないでより古いバージョンのファイルを検索することができないことのような問題がないわけではない。しかしながら、この解決策はテープだけのバックアップ解決策を使用して容易にアドレス指定できないいくつかの主要な障害を解決する。

【００１２】ネットワーク上の大部分のワークステーションは、同一の内容を有する多数のファイル、特に、オペレーティングシステムファイル、プログラムファイル、及びソフトウェアの一部として分配され、ユーザのディスクに記憶され、決して変更されない他のファイルを所有していることは容易に確認される。このような共通ファイルによって占有されたディスク内容の割合は、時間とともに、特にディスク駆動容量が増大し、より多くのソフトウェアがCD - ROMで配給されるつれて増加することは本当であるようにも思える。しかしながら、前述の従来のバックアップ方式のどれもいずれにしてもこれらの現象を利用していないことを確認すべきである。

【００１３】

【課題を解決するための手段】本発明の目的は、コンピュータネットワーク上の複数のノードからのデータをバックアップすることに歴史的に関連する問題の多くを解決することにある。従来の技術に対比して、本発明は、同時にネットワーク帯域幅消費を減少し、バックアップ及び復旧に必要な時間を減少し、中央管理を可能にし、ユーザワークステーションでのバックアップ処理を自動化し、いかなる管理者の介入もなしに以前のファイルの全てのバージョンへのアクセスを提供し、そしてユーザが自分自身のアプリケーションを使用して直接バックアップからファイルへのアクセスを可能にする低コストバックアップ解決策を提供する。

【００１４】本発明では、ファイルは、コンピュータネットワークの複数のノード上のディスクボリュームから共通ランダムアクセスバックアップ記憶手段、共通のディスクボリュームへバックアップされる。バックアップ

は、各ノードに対して自動的に及び独立して生じるようにユーザ又はバックアップ管理者のいずれかによってスケジュールすることができる。バックアップ処理の一部として、複写ファイル（又はファイルの部分）はノードにわたって識別することができるので、複写ファイル（又はそれの部分）の内容の１つのコピーだけがバックアップ記憶手段に記憶される。好ましい実施例は、何百万のファイルがバックアップシステムに付加された場合さえ、ネットワーク帯域幅のその使用に非常に効果的である複写ファイルを識別する探索方法を含んでいる。特定のボリュームに関する初期バックアップ後の各バックアップ動作の場合、前のバックアップ以来変更されているこれらのファイルだけがボリュームから読み出され、バックアップ記憶手段に記憶される必要がある。すなわち、未修正ファイルの内容に対するポインタはバックアップのためのディレクトリ情報とともに記憶されている。さらに、ファイルと前のバックアップにおけるそのバージョンとの差は、このファイルに対する変更だけがバックアップ記憶手段に書き込まれる必要があり、バックアップ記憶手段に書き込まれたほとんど全てのデータは無損失圧縮アルゴリズムを使用して圧縮される。これらの“データリダクション”エンハンスメントは、バックアップを実行するのに必要な記憶量及びネットワーク帯域幅量の両方を著しく減少させる。実際、データリダクションは、大抵の場合、必要とされる記憶量を、バックアップ記憶手段としてディスク駆動装置を使用するためのシステムコストが、特にディスク記憶装置の急速に下落しているコストを与えられたこのような環境の中での従来のテープバックアップシステムのコストよりもより安価である点まで低下させるのに十分有効である。

【００１５】バックアップデータが公然とアクセスできる共有ファイルサーバに記憶されている場合さえ、データ機密は、ファイル内容のハッシュ関数から生成された暗号化キーを使用して各ファイルの内容を暗号化することによって保持することができるので、ファイルのコピーを１度バックアップしたユーザだけは暗号化キーを作成でき、ファイル内容にアクセスできる。

【００１６】バックアップからファイルを見るか又は復旧するために、ユーザは、バックアップの時点で全部の元のディスクボリュームのディレクトリ構造と同一のディレクトリ構造を有する実時間（すなわち、ディスクアクセス時間を有し、テープアクセス時間を有しない）一時ディスクボリュームのようなバックアップセットを備えることができる。したがって、ユーザは、個別の復旧プログラムを使用してファイルを最初にコピーする必要がなく自分自身のアプリケーションを使用して直接にファイルをアクセスできる。バックアップディスクボリュームは読みだし専用モードで備えることができる。すなわち、その代わりに、ファイルに対する一時的修正を可能にするような書き込みアクセスを実現できる。ただし、一旦バックアップボリュームが取り外されると、全てのこのような修正は通常消失される。

【００１７】

【発明の好ましい実施の形態】本発明の好ましい実施例は、そのバックアップ記憶手段としてネットワークファイルサーバ上のディスク領域を使用する。各クライアントワークステーションは、ファイルサーバ上の予め割り当てられたロケーション又はディレクトリにバックアップデータをコピーし、並びにユーザにわたる複写ファイルを識別し、ファイルバージョン間の差（すなわちデルタ）を計算するためにバックアップ“データベース”を探索する責任がある。したがって、性能及びセキュリティにきわめて重要なある種のハウスキーピング機能は、ファイルサーバそれ自体を含むいかなるネットワークノード上で実行できるエージェントタスクによって実行される必要があるけれども、好ましい実施例は、正確にはクライアント／サーバシステムではない。

【００１８】他の実施例では、バックアップ記憶手段は、アプリケーションサーバ（バックアップサーバ）上のディスク領域からなる。ネットワークノードは、伝統的なクライアント／サーバパラダイムでバックアップサーバとつながっている。エージェント機能はバックアップサーバによって実行される。この実施例は、好ましい実施例よりもわずかに高いセキュリティを実現できるが、個別のサーバを要求するために通常より多くのコストがかかる。ただし、バックアップサーバが他のアプリケーションサービスを実現できるならば、このコストをある程度償却することが可能であることもある。このような実施例は、この方式の評価に影響を及ぼすかもしれない計算負荷（例えば、複写ファイルを識別する）がサーバに集中する傾向もある。ただし、所望ならば、当業者にすぐに明かになるクライアントノードにわたる計算負荷のより多くを分配するための簡単な方法がある。本発明の範囲内にあるファイルサーバ方式及びアプリケーションサーバ方式のいろいろな特徴の混ざり合ったものからなる多数の他の可能な実施例もある。

【００１９】さらに他の実施例では、バックアップ記憶手段は、長い間アクセスされなかったファイルがディスクからテープ又は光ディスクのような二次記憶手段に移動される階層記憶管理（HSM）を組み込んでいる。HSMの主目的は、いくつかのファイルをアクセスする際の付加遅延を除いて、移動がシステムにトランスペアレントであることを可能にする管理ツールを備えることによって非常に大きい記憶システムのための記憶コストを節約することにある。本発明のバックアップ記憶手段とともにいかなる形式のHSMの使用もここで論議された概念のどれにも著しい影響を及ぼさない。しかしながら、二次記憶をアクセスする際に被る遅延はシステムを大いに使えなくするので、バックアップ動作及び復旧動作の性能をそこなわないように注意しなければならない。確かに、

13

バックアップ性能に悪影響を及ぼさないで二次記憶に移動することができる本発明のバックアップデータ及びディレクトリファイルの内容の部分を識別することはかなり簡単である。幸に、大抵の場合、本発明のデータリダクション法は、HSMを使用しない場合さえディスク記憶コストを許容レベルに抑えるのに十分強力である。

【００２０】１．バックアップ処理

図１に示されるように、好ましい実施例では、バックアップされるノードは、ワークステーション102、デスクトップパーソナルコンピュータ103、ラップトップコンピュータ104、又はネットワーク上の他のサーバ105のいずれであってもよい。全ての通信は、バックアップ記憶装置101にネットワーク106を通じてファイルを作成又は修正することによって達成される。図２に示されるように、各ノードは２つのディレクトリ、すなわちユーザディレクトリ及びネットワークファイルサーバ100のディスクボリュームの中に含まれるバックアップ記憶手段101上のシステムディレクトリを割り当てられている。ノードは、バックアップデータを示すそのユーザディレクトリ（例えば、図２の＼BACKUP＼USERS＼USER２、125）へのネットワーク書き込みアクセスを有する。バックアップ管理者は、製品の一部として提供された管理者ソフトウェア機能を使用してバックアップシステムを構成する。バックアップ管理者によって選択されたネットワークノード107上で実行するバックアップエージェント処理108は、そのバックアップファイルをシステムディレクトリ（例えば、図２の＼BACKUP＼SYSTEM＼USER２、128）に移動する。このシステムディレクトリは、全てのノード（エージェント108を除く）に対して読み出しだけをするために割り当てられたネットワーク権を有するので、いかなるユーザノードも故意に又は誤って移動バックアップデータを破損することができない。２つのディレクトリを使用することは厳密には必要ないが、データ一貫性はこの方式を使用する好ましい実施例の共有ファイル環境で著しく改良される。

【００２１】好ましい実施例では、エージェント108は、図２に示された全てのディレクトリへの読み出し一書き込みアクセスを有する。各ユーザは、BACKUP＼SYSTEM 122の下の全てのディレクトリに対し読み出し専用アクセスを与えられているが、ユーザは、読み出し一書き込みアクセスを有する自分自身のディレクトリ（例えば、図２の＼BACKUP＼USERS＼USER２、125）以外のBACKUP＼USERS 121の下のディレクトリのどれにもアクセスできない。いかなるユーザもバックアップファイルが移動される前に他のユーザの示されたバックアップファイルを破損することができないので、このように示されたディレクトリへのアクセスを制限することはセキュリティをさらに増加する。しかしながら、好ましい実施例では、全てのバックアップファイルは、暗号化され、破損を検出するために使用することができるチェックサムを

14

有するので、かなり行儀の良いユーザであると仮定すると、セキュリティを著しく弱めないでBACKUP＼USERS 121の下の全てのディレクトリへの読み出し一書き込みアクセスを全てのユーザに与えることも、他の実施例において可能である（そして多分ネットワーク管理者の見地からより容易である）。エージェント108は、各バックアップファイルを移動している間、各バックアップファイルの一貫性をチェックする。すなわち、いかなるエラーも検出されるならば、ファイルは移動されなく、したがって、BACKUP＼SYSTEMディレクトリのデータの一貫性を維持する。ネットワークアクセス権及びエージェント108を使用する好ましい実施例のこの一般的な方式は、従来の各クライアントノードが共有ファイルへの完全な読み出し一書き込みアクセスを有し、したがって、この共有ファイルは非常に破損されやすい共有ファイルアプリケーションにおけるよりもさらに高いレベルのデータセキュリティ及びデータ一貫性を生じることを確認すべきである。

【００２２】図３は、好ましい実施例のバックアップシステムの一部として作成された主なタイプのファイル（並びにその相互関係のいくつか）を示す。ノード上のディスクボリュームのバックアップ中、好ましい実施例のバックアップ処理は、ソースディスクボリューム上の全てのファイルを４つのカテゴリ、すなわち、新規ファイル、未変更ファイル、更新済ファイル及び修正済ファイルに分ける。新規ファイルは、前のバックアップの時点で同一ディレクトリ上に存在しないファイルである。未変更ファイルは、最後のバックアップの時点で存在し、その時点以来変更されていない（例えば、これらのファイルは同一の時間、日付、及びサイズをなお有する）。更新済ファイルは、前のバックアップの時点からNu日（ここで、Nuはユーザが選択できるオプションである(一般的には、14～90日の範囲内)）以上の間、変更されないが、最後のバックアップ以来変更されているファイルである。全ての他のファイルは修正済として分類される。所与のボリュームの最初のバックアップが実行されると、全てのファイルは新規として分類される。各新規又は更新済ファイルに関して、バックアップソフトウェアは、マッチングファイルに対してグローバルディレクトリデータベース145を端から端まで探索する。グローバルディレクトリデータベース145は、ディレクトリ＼BACKUP＼SYSTEM＼GLOBAL127の中のエージェント処理108によって作成され、保持される。エージェント108がバックアップセットを＼BACKUP＼USERSパス121から＼BACKUP＼SYSTEMパス122に移動する度に、エージェント108は、バックアップセットにおける新規ファイル及び更新済ファイルを探索し、これらのファイルをグローバルディレクトリデータベース145に付加する。照合ファイルがデータベースで得られるならば、そのファイルの内容への参照が、後述されるようにファイルデータそのも

15

のの代わりに記憶される。同様に、未変更ファイルに関しては、前のファイル内容への参照だけが記憶される。

【００２３】探索時間及び帯域幅を最少にするために、修正済ファイルに対してはグローバルディレクトリデータベース145の端から端までの探索を行わないことが好ましいと信じられている。同じ理由のために、かつデータベースの増大を最少にするために、修正済ファイルはグローバルディレクトリデータベース145に追加されない。その代わりに、修正済のファイルの内容は、ファイルの最新バージョンからの差を計算し、どちらがより小さいにしても、そっくりそのまま差又は新しいバージョンのいずれかを保存することによってバックアップの中に記憶する。この差は、当業者に公知であるいかなる方法でも計算することができ、表示することができる。修正済カテゴリの特別のユーザ定義サブセットとみなすことができる更新済カテゴリは、ユーザにわたるたまにしか更新されない複写ファイルを識別するのに役立つ。このようなファイルの一つの共通例は、ワードプロセッサ又はある種の他の一般に普及しているアプリケーションの実行可能なものの新規バージョンである。Nuをゼロに設定することは修正済カテゴリを除外する(すなわち、全ての変更済ファイルは更新済カテゴリにある）のに対して、Nuを無限大に設定することは更新済カテゴリを除外する（すなわち、全ての変更済ファイルは修正済カテゴリにある）ことに注目すべきである。

【００２４】１．１．バックアップディレクトリファイル

好ましい実施例のバックアップ処理は、各バックアップセット、すなわち、バックアップディレクトリファイル（例えば、143）及びバックアップデータファイル（例えば、144 ）についての情報を含む２つのファイルを実際に作成する。他の実施例では、これらのファイルを組み合わせると単一のファイルができる。バックアップディレクトリファイルの内容は、ソースディスクボリュームのディレクトリ構造並びに各ファイルのためのデータがどこで得られるべきであるかを指示するバックアップデータファイル（例えば、144、149、及び他のユーザのバックアップデータファイル）の中のポインタを示す。本発明の一つの主要な特徴は、ユーザにわたる複写情報を参照することを含む、情報そのものを複写する代わりにデータ及びディレクトリ情報を示すポインタを複写することによって達成されたデータリダクションである。本発明においてデータリダクションを達成する際のバックアップディレクトリファイルの役割を説明するために、DOSディスクボリュームに対するバックアップディレクトリファイル（例えば、143）の主要部の記述は、周知の形式言語技術（例えば、1976年発行、ニクラウス・ウルス（Nicklaus Wirth)著の「アルゴリズム＋データ構造＝プログラム」、第281～291ページを参照）であるバッカスーナウア記法(BNF)で図４に示されている。

16

図４の内容を論議する前に、著者毎に構文においてわずかな変化があるので、我々は、自分達のBNF記法を明確に定義する。非端末はカギカッコ（ギュメ）で囲まれる（例えば、＜fileEntry＞)。:=シンボルは、形式定義を示している。端末は、単一２進数字（０又は１）又はCのような構文、すなわち、８ビットバイトに対しては0xUU、16ビットワードに対しては0xUUUU、及び32ビットワードに対しては0xUUUUUUUUを使用する16進数量として示される。端末値の値域は、間に２つのピリオドを有する２つの端末数量、例えば、0x00..0xFEとして示される。｜文字は、“一方又は他方”を示すメタシンボルであるのに対して、角括弧 [] は任意のフィールドを示し、アスタリスク（＊）は１つ以上のフィールドの反復を示す。したがって、例えば、［＜externDirItem＞］＊は、ゼロ以上の非端末＜externDirItem＞を示す。ダブルスラッシュ//は、行の終わりに対するコメントを示す。

【００２５】図４は、バックアップディレクトリファイル（例えば、143）のディレクトリ情報のフォーマットを示す。200で、ファイルの＜volumeDirInfo＞部は、一連の＜subdirFileList＞レコード201であるように定義される。この＜subdirFileList＞レコード201の後には、＜externDirItem＞レコード220の別個のリストが続く。各＜subdirFileList＞レコード201は、ファイルのためのディレクトリエントリ及び単一ディレクトリ内のサブディレクトリを含んでいる。特に、201で示されるように、各＜subdirFileList＞レコードは、一連の＜fileEntry＞レコード及び＜subdirEntry＞レコード207、208（関連ディレクトリの中でそれぞれ得られるファイル及びサブディレクトリのためのディレクトリエントリを含む）からなり、＜endOfList＞マーカ202(例えば、0バイト)によって終了される。＜endOfList＞マーカ202の後には、このディレクトリに関連した＜externDirItem＞レコード220の数を表す可変長符号化整数＜itemCount＞204である＜externCount＞203が続く。好ましい実施例で使用される＜itemCount＞の特定の符号化(204、205、206)は重要でない。すなわち、平均符号サイズを最小にするためには、小さいカウントの方が大きなカウントよりもより一層一般的であるという事実を利用することは、通常、符号化のために望ましいけれども、多数の簡単な他の符号化は、同様に十分な働きをする。他の実施例では、各＜subdirFileList＞レコード201に関連した＜externDirItem＞レコード220は、200で示されるように、別々のセクション内の代わりに＜externCount＞フィールド203の直後に記憶することができる。すなわち、好ましい実施例では、軽度の最適化としてこれらのセクションを別々にしておき、どの外部項目が＜subdirFileList＞セクション201を構文解析するためのオーバーヘッドなしに参照されるかを調べるため、エージェント108が全＜externDirItem＞リスト220の端から端まで

迅速に走査することを可能にする。

【００２６】好ましい実施例では、ディレクトリツリーは、従来の深さ優先配列で＜subdirFileList＞レコード201を配置することによって暗黙に示される。換言すれば、ディスク上のディレクトリツリーを“走査”している間にサブディレクトリに出会う度に、そのサブディレクトリのための＜subdirEntry＞レコード208は、現＜subdirFileList＞201に付加され、そのサブディレクトリを表すトークンは一時内部スタック上に押される。現サブディレクトリの処理が完了すると、＜endOfList＞フィールド202及び＜externCount＞フィールド203が、201に示されるように付加され、次に、処理は内部スタックをポップオフするトークンによって示されたサブディレクトリにおいて継続する。スタックが空であるならば、全ディレクトリツリーは完成する。他の実施例では、他のツリー配列（例えば、深さ優先）は同様な結果を達成するために使用することができる。

【００２７】各明示＜fileEntry＞レコード207は、そのユーザにとって全てのバックアップセットにわたってユニークなディレクトリ項目番号(＜dirItemNum＞223)を割り当てられる。好ましい実施例では、この番号は図４の223で示されるように増分31ビット数量である。すなわち、このサイズは、例えば、オーバーフローが発生する前に50年以上の間、１日当たり各々が10,000の変更済ファイルを有する10のバックアップまでに対して十分である。もちろん、必要ならば、より多量のビットが使用することができる。使用される＜dirItemNum＞値223の値域は、バックアップディレクトリファイル（例えば、143）の他の場所に定義されている。すなわち、空間を節約するために、＜volumeDirInfo＞レコード200における各＜fileEntry＞207は、値域内で次の＜dirItemNum＞223を暗黙に割り当てられるので、いかなる明示＜dirItemNum＞223も各＜fileEntry＞207とともに記憶される必要がない。バックアップ処理中、未変更ファイルが見つかると、そのファイルのための＜fileEntry＞207を複写する代わりに、参照は、現ディレクトリに関連した＜externDirItem＞220リスト中にその＜dirItemNum＞223を含むことによって前の＜fileEntry＞207に付加されてもよい。複数の未変更ファイルが連続＜dirItemNum＞値223とともに見つかるかなり一般的な場合、空間を節約するために、このシーケンスは、例えば、１ビットタグ（１）、31ビットの＜dirItemNum＞223、及び参照される連続外部＜fileEntry＞レコード207の数を示す＜itemCount＞レコード204からなる＜manyItems＞レコード222によって示される。さもなくば、＜externDirItem＞220は、このフィールド222と＜manyItems＞フィールドとを識別する１ビットタグ(0)からなる＜oneItem＞221として示され、参照＜fileEntry＞207の31ビットの＜dirItemNum＞223が続く。好ましい実施例では、＜externCount＞フィールド203は、＜externDirItem＞220レコードの

数を計数し、それによって参照された＜fileEntry＞レコード207の全数を計数するのではないことに注目すべきである。

【００２８】＜fileEntry＞レコード及び＜subdirEntry＞レコードは、可変長のものであり、図４の207及び208で示されるようにいくつかのフィールドからなる。これらのフィールドは、基礎をなすファイルシステムの属性によって左右される。説明のために、図４の定義は、DOS FATファイルシステムに必要な属性を含むが、異なるファイルシステム（例えば、マッキントッシュ、OS/2 HPFF、ネットウェア等）における異なる属性を考慮するために＜fileEntry＞定義207に対して明かな修正を加えることができる。好ましい実施例では、バックアップディレクトリファイル（例えば、143）のヘッダは、ソースファイルシステムを指定するフィールドを含み、したがって、このバックアップの中の＜fileEntry＞レコード207の特定のフォーマットを示す。図４の例では、第１のフィールドは、ファイル名を示すゼロ終了可変長文字ストリング（219で規定されているように、＜asciiz＞）である212で定義された＜fileName＞レコードである。次に、読み出し専用、ディレクトリ、隠し、システム等のような属性ビットを含む単一バイトである＜fileAttrib＞フィールド209が来る。ファイル修正時間＜fileTime＞210が続く。すなわち、この32ビット数量は、ファイルが最後に修正されるときの時間および日付の両方を含む。より進歩したファイルシステムでは、最後のアクセス時間、作成時間等のようないくつかの他の時間値がここで加えることができる。＜fileSize＞フィールド211は、数バイトのファイルのサイズを示す32ビット数量である。最後に、＜fileID＞214フィールドは、このファイルと関連するファイルデータがどこで見つけることができるかを示す。214に示されるように、この情報は、＜userIndex＞216及び＜fileIndex＞215を含んでいる。バックアップシステムの各ユーザは、好ましい実施例では16ビット数量であるユニークなユーザ番号＜userIndex＞216を有する。同様に、各ファイルは、ディレクトリ項目に対して＜dirItemNum＞223と同様にユニークな番号を割り当てられる。すなわち、この＜fileIndex＞215は、好ましい実施例では、32ビット数量である。＜fileID＞214を形成する２つのフィールドは、後述されるように、ファイルデータを含む適切なバックアップデータファイル（例えば、148）を見つけるために使用することができる。＜subdirEntry＞レコード208は、＜fileTime＞フィールド210がディレクトリ作成時間を示す以外、＜fileEntry＞レコード207と同じ最初の３つのフィールドからなる。他の実施例では、各＜fileEntry＞レコード207は、ファイルの前のバージョンに対して＜fileEntry＞207を直接参照する＜dirItemNum＞223である＜lastVersion＞フィールドも含んでいる。すなわち、この技術は、全てのバックアップディレクトリファ

19

イル（ファイルが変更されない場合、バックアップディレクトリファイルを含む）を読み、構文解析することによってかなりもっとゆっくりと再構成することができるファイルの全ての独特なバージョンのリンクリストを実現できる。

【００２９】好ましい実施例では、前のバックアップから未変更＜subdirEntry＞レコード208を参照する方法がない。換言すれば、これらのディレクトリ内のファイルが前述のように＜externDirItem＞セクション220内で参照することによって組み込むことができるけれども、ディレクトリの全てのトリーは、各バックアップディレクトリファイル（例えば、143）で明確に表示されねばならない。好ましい実施例ではこのいくらかの任意の決定は、バックアップを簡単にし、いくつかのバックアップディレクトリファイルのサイズにおいて少ないコストでロジックを復旧するようになされるが、他の実施例では、未変更サブディレクトリ（及び全ファイルツリー／サブディレクトリツリー）を参照することは簡単である。バックアップディレクトリファイル（例えば、143）のサイズは、通常、バックアップデータファイル（例えば、144）の何分の一かのサイズであり、バックアップディレクトリファイルのサイズに対してサブディレクトリエントリだけからの寄与は、通常重要でないので、この問題はせいぜい非常に小さい関心事項であるように思える。

【００３０】多少関連し、恐らくより大きい関心は、図４の定義によれば、所与のバックアップディレクトリファイル（例えば、143）によって参照される外部のバックアップディレクトリファイル（例えば、148）の数に関する制限はないという事実にある。万一この数が制限なしに増えるなら、復旧の実行時、ディレクトリツリーを再構成するのに必要な時間量は、たとえ全てのバックアップディレクトリファイルがディスク上にあるとしても、かなり大きいことがある。実際には、好ましい実施例では、バックアップ時、バックアップディレクトリファイル(例えば、143）を構成中に参照することができる外部バックアップディレクトリファイル（例えば、148）の数N_Dに対する制限がある。一般的には、この数は、N_D＝５－20ファイルの値域内に設定される。この結果、未変更ファイル用の＜fileEntry＞207レコードが、ほぼ全てのN_Dのバックアップについて明確に再び含められることになる。この処理は、バックアップ記憶101の全記憶条件をわずかに増加させるが、復旧動作中、適度な応答時間を保証する。

【００３１】好ましい実施例では、各ユーザのバックアップファイルは、自分自身の前のバックアップから＜externDirItem＞レコード220を参照できるだけで、他のユーザの他のバックアップから＜externDirItem＞レコード220を参照できない。全記憶条件において非常に少ないコストですむこの決定は、全てのユーザディレクトリ

20

情報の機密を保持したい願いから生じる。次に分かるように、バックアップ・ファイルの内容は暗号化されているので、誰しも、ファイルの内容をアクセスするまでもなく、これらを知ることでプライバシーが損なわれる可能性のある、他人のファイルの、ファイル名、ファイル・サイズ、ファイル生成日付、あるいは、ファイル属性を知ることができない。それに反して、バックアップされるファイルのデータ内容は、後述のユニークな暗号化キープロトコルにより保証される機密をユーザ間で共有することができる。バックアップディレクトリファイルのサイズが他の実施例では重要な問題になる（例えば、新しい種類のファイルシステム）ならば、データのために使用される技術と同様な技術は、所望ならば、領域を節約するようにディレクトリエントリに適用することができる。しかしながら、この時点で好ましい実施例に関心のあるファイルシステムに関しては、バックアップディレクトリファイルのサイズをさらに最小化する強制的な要求はないように思える。

【００３２】図４のBNF定義の意味をさらに図示するために、図５で、＜volumeDirInfo＞レコード200のフォーマットのいくつかの例を示す。フォーマットは、可変長フィールドの非常にフレキシブルな（多少プリミティブならば）出力を考慮する8086アセンブリ言語のフォーマットである。セミコロン（；）は、行の終わりのためのコメントとして役立つ。下記の指示文は出力を制御するために使用される。すなわち、

db=emit ８ビットバイト
dw=emit 16ビットワード
dd=emit 32ビットdワード

16進法定数は、’H’（例えば、80000000H）で終わる。この例のいくつかのフィールドは、（？）表示を使用して未定義のままである。すなわち、例えば、特定の時間は、本説明にとっての関心事項でないため、ファイル時間／日付は未指定である。一般に、使用方法を明かにするために、各行に、その行に対応するBNF非端末を含むコメントが続く。行毎のコメントは、上記に詳述されている図４のBNFを直接に参照する。356で開始する＜externDirItem＞リストは、参照された＜fileEntry＞レコードのどんな実際の内容も指示しない。ディレクトリエントリを得るために別々に参照されたバックアップディレクトリファイルの内容を読み出し、構文解析することは必要である。

【００３３】好ましい実施例では、各バックアップディレクトリファイル（例えば、143）は、いくつかの他のセクションを含んでいる。これらのセクションは、ここでは簡単に記載されている。これらは、通常、多くの他のバックアップ製品で使用されているよく理解された技術を必要とし、したがって当業者によって容易に理解される。しかしながら、本発明に対してより幅広い背景状況を与えるためにこれらの他のセクションの内容及び目

的の簡単な説明をすることは有用である。各セクションは、破損検査を考慮するためのチェックサム又はCRCによって保護され、（<volumeDirInfo>セクション200を含んでいる）セクションの中の多数は、米国特許第5,016,009号、又は米国特許出願第07/927,343号（1992年８月10日出願、発明の名称が、「一致文字列探索及びハフマン符号化を使用するデータ圧縮装置及び方法」である）に記載されている圧縮技術のような周知の圧縮技術である。上記特許の両者とも、本発明の譲受人に譲渡され、参照することによってここに組み込まれている。さらに、（ヘッダ以外の）各セクションは、データ暗号化規格(DES)又はRSAの周知のRC2アルゴリズム又はRC4アルゴリズムのような専用キー暗号化方式を使用して暗号化される。この暗号化に対するキー管理プロトコルは下記に詳述される。最後に、ある種のプリミティブエラー訂正能力は、ファイル内容の終わりに全パリティセクタの中の一部を付加することによって各ファイルに組み込まれる。

【００３４】好ましい実施例の各バックアップディレクトリファイルは、まず、シグナチャ及び作成時刻記録、さらに、ファイルフォーマットバージョン関する情報、ファイルサイズ、及び全ての他のセクションのロケーション及びサイズを識別するポインタとを含んでいるヘッダから始まる。“bkupDescription”セクションは、ユーザ生成注釈文字列、バックアップの時間、新しいファイル及びバイトのカウント、生成された新しい<dirItemNum>及び<fileIndex>レコード223、215の値域、バックアップファイル数、<userIndex>216、及びバックアップに対する資源ボリュームの指定を含むバックアップ動作に関する記述情報を含んでいる。<dirIndexRange>セクションは、前述のように、<dirItemNum>223割り当てが<volumeDirInfo>200において行われるファイルに含まれる厳密な新しい<dirItemNum>223値のセットを識別する小さい可変長レコードである。すなわち、通常、単一の隣接する値域の値だけが存在するが、エージェント108が（後述の）串刺し（consolidation）演算を実行した後、複数の非隣接値域が単一ファイル内に存在することは可能である。<dirItemPtr>セクションは、<volumeDirInfo>200の中へのポインタのアレイ、すなわち各<fileEntry>207に対して1つのポインタを含んでいる。このセクションは冗長であり、<volumeDirInfo>200を<dirIndexRange>セクションとともに、構文解析することよって再構成することができ、<dirItemNum>参照223を介して別のバックアップディレクトリファイルから<fileEntry>レコード207へのアクセスを速くするのに役立つ。最後に、“fileDecryptKey”セクションは、データファイル内容の復号化のために使用される専用暗号化キー（例えば、DESのために）を含んでいる。このセクション内には、<volumeDirInfo>200内の各<fileEntry>207に対する１つのキーがある。実際、概念的にはこのキーは<fileEntry>レコード207の一部であるが、<fileEntry>207内に直接キーを含むことは単に<volumeDirInfo>セクション200の圧縮比を低下するので、好ましい実施例では、本キーは別のセクション内にある。

【００３５】同様な結果を得るためにバックアップディレクトリファイル内に情報を構成するには多くの匹敵する方法がある。ここに記載した好ましい実施例の特定のレコードフォーマットは、本発明の範囲を限定することを意図していない。

【００３６】１．２．バックアップデータファイル

バックアップデータファイル（例えば、144）は、バックアップセット内に含まれるファイルからのデータを含んでいる。このデータのいくつかは、本ユーザ又は他のユーザのいずれから前のバックアップ（例えば、149）からの他のバックアップデータファイルの中を参照することによって示すことができる。バックアップデータファイル内に含まれる各々の独自ファイルは、好ましい実施例では32ビット数であり、そのファイルを参照するために使用される<fileIndex>215を割り当てられる。<dirItemNum>223値と<fileIndex>215値との間にはいかなる１対１対の関係もないことを確認せよ。例えば、ユーザＡがユーザＢによって既にバックアップされたファイルの正確なコピーを有しているならば、ユーザＡの<fileEntry>207は、ユーザＢの<fileEntry>207と同様に同一の<fileID>214（すなわち、<fileIndex>215及び<userIndex>216）を含むが、それらは、前述のように、好ましい実施例ではユーザ間では共有されない別個の<dirItemNum>値223を有する。バックアップデータファイル内のデータの大部分は圧縮され、バックアップセット内に含まれる各ファイルからのデータは、専用ユーザキーの代わりにファイル指定キー（143のようなバックアップディレクトリファイルの“fileDecryptKey”セクション内に記憶されている）を使用して暗号化される。換言すれば、多数の暗号化キーは、通常各バックアップデータファイル内で使用される。データ機密を保証するために使用されるキー管理プロトコルは、下記に詳細に説明されるが、ネット結果は、好ましい実施例では、各バックアップデータファイルの内容が専用ユーザ指定キーで暗号化されるバックアップディレクトリファイルの内容と比較すれば、有効に公に利用可能であるということである。

【００３７】好ましい実施例の高レベルのバックアップデータファイル（例えば、144）の配置のブロック図が図６に示されている。このファイルは、４つの主セクションからなる。その中のデータブロックセクション161は、バックアップセット内に含まれるファイルの実際の内容を含んでいるので、一般的には断然最も大きい。その大きさのために、好ましい実施例では、データブロックセクション161は、ファイルデータがデータブロック

23

を再度移動させる必要なしにバックアップファイルデータに直接書き込まれるように一定のサイズのヘッダ160の直後にくる。ヘッダセクション160は、シグナチャ及び作成時刻記録、並びにファイルフォーマットバージョンに関する情報、ファイルサイズ、及びFileInfotrsセクション178を示すポインタを含んでいる。バックアップディレクトリファイルと同様に、バックアップデータファイルは、小さいディスク傷がバックアップ記憶手段101のセクタ上に発生する場合に簡単なエラー補正を可能にするためのパリティセクタも含むことができる。FileInfotrsセクション178は、バックアップデータファイルに示された正確な＜fileIndex＞値215のセットを示す可変サイズレコードを含んでいる。すなわち、このレコードは、前述のバックアップディレクトリファイルの“dirIndexRange”セクションに非常に類似しており、一般的には、単一隣接値域の値だけからなる。FileInfotrsセクション178の残りは、一定サイズのエントリのアレイ（例えば、181）、すなわち値域内に示される＜fileIndex＞値215当たり１エントリを含んでいる。各エントリは、ファイル当たり１つの可変サイズエントリ（例えば、176）があるFileInfoセクション175を示すポインタ（例えば、179）を含んでいる。さらに、FileInfotrsセクション内の各エントリ（例えば、181）は、各新規又は更新済ファイルをグローバルディレクトリデータベース145の中に入れるのに必要な他のファイル指定情報（例えば、ファイル内容の初期ブロックにわたるファイルサイズ及びCRC）を含んでいる。各FileInfoエントリ（例えば、176）は、DataBlockセクション161又は他のバックアップデータファイル内に含まれるファイルの内容を示す可変長のポインタアレイを含むファイルの内容に関する情報を含んでいる。

【００３８】例えば、図６は、バックアップセット内に含まれる２つのファイルのいくつかの細部を示している。ファイルＡに対するFileInfoPtrエントリ181は、ファイルＡに対するFileInfoエントリ176を示すポインタ179を含んでいる。このエントリ176は、データブロック163及び165を示すポインタ164及び166をDataBlockセクション161の中にそれぞれ含むポインタのセット173を含む。同様に、他のバックアップデータファイル内にデータブロックを示すポインタ167を含んでいる。ファイルＡに関連するこのバックアップデータファイル内の全てのデータブロック（163及び165を含む）は、168で示されるようなファイルＡに対する暗号化キーを使用して暗号化される。このキーは、その＜fileID＞214がファイルＡを参照する＜fileEntry＞207を含むバックアップディレクトリファイルの“fileDecryptKey”セクション内に記憶されている。同様に、ファイルＢに対するFileInfoPtrエントリ182は、ファイルＢに対するFileInfoエントリ177のためのポインタ180を含んでいる。このエントリ177は、データブロック171を示すポインタ170をDataBlockセクション161内に含むポインタのセット174を含む。同様に、他のバックアップデータファイル内にデータブロックを示すポインタ172を含んでいる。ファイルＢに関連するこのバックアップデータファイル内の全てのデータブロック（171を含む）は、169で示されるようなファイルＢに対する暗号化キーを使用して暗号化される。

24

【００３９】＜fileID＞214及び復号化キーが与えられれば、ファイルを“復旧”するためにファイル内容を抽出するのは比較的簡単なことである。最初に、関与する＜fileIndex＞215を含むバックアップデータファイル用に、＜userIndex＞216によって識別されたユーザのバックアップデータファイルの端から端まで探索が実行される。＜fileIndex＞値215の値域を含んでいるヘッダ160及びFileInfoPtrs178が暗号化されていないために、この探索は容易に実行することができる。好ましい実施例では、＼BACKUP＼USERS121から＼BACKUP＼SYSTEM122へのバックアップデータファイルの移動処理の一部として、エージェント108が特別のインデックス値域ルックアップファイル（例えば、図３の151）を形成するので、探索は、通常、さらに迅速に実行することができる。バックアップデータ及びディレクトリファイルの内容から常に再形成することができるという意味で余分であるこのファイルは、インデックス値域をバックアップデータファイル名に写像し、高速２進探索のために仕組まれているテーブルを含んでいる。識別された適当なバックアップデータファイルに関しては、このファイルが開かれ、FileInfoPtrsセクションのためのポインタ162がヘッダ160から読み出される。次に、FileInfoPtrsセクション161のインデックス値域レコードは走査され、どのポインタが所与の＜fileIndex＞215に一致しているかを識別する。したがって、そのポインタは関与するファイルに対するFileInfoエントリ（例えば、176）をインデックスするために使用される。ポインタ（例えば、173）は、このバックアップデータファイル（例えば、163）又は“外部”バックアップデータファイル（例えば、149）内のいずれかに存在し得る、関与する、ファイルの各部に対応するデータブロックを示すFileInfoエントリから得られる。次に、これらのブロックは、読み出され、復号化され、そして解凍され、元のファイル内容を表示する。ファイル内容のいかなる部分のアクセスも少数のディスクアクセスのみを必要とすることがかなり容易に分かる。アクセス回数は“通常”のファイルシステムのファイルのアクセスに要する回数よりも多分多いけれども、復旧中のアクセス時間の単位は、数ミリ秒（最悪の状態でも数十分の一秒）で測定され、従来のテープバックアップからの復旧操作に通常要する数十秒又は数十分に比べ十分小さい。明かに、復旧“アクセス”時間を最適にするために、復旧ソフトウェアは、バックアップディレクトリ及びバックアップデータファイルの

内容のためのインテリジェントキャッシングアルゴリズムを含んでいる。

【００４０】この高レベルのバックアップデータファイルのいろいろなセクションを理解するならば、これらのセクションのいくつかのフォーマットに関するかなり多くの細部を示すBNF定義セットを含む図７を考察すべきである。400で、全<bkupDataFile>は、上記に４つの主セクション（<header>セクション401、<dataBlock>セクション405、<fileInfo>セクション408、及び<fInfoPtrs>セクション432）からなるように定義されている。図７の残りはこれらのセクションの内容を述べる。

【００４１】<header>セクションの関連部分は401に列挙されている。特に、<fInfoPtrOffset>フィールドは、<fInfoPtrs>セクション432の開始のポインタ（好ましい実施例では32ビット）として402で規定されている。<indexRangeCnt>フィールドは、<fInfoPtrs>セクション432内の<indexRange>エントリ433の数のカウントとして403で規定されている。

【００４２】405及び406で、各<dataBlock>は８ビットバイトの可変長アレイであると規定されている。好ましい実施例では、各<dataBlock>405は、４の倍数であるバックアップデータファイル（例えば、144）内のオフセットで始まるので、オフセットは30ビット単位で符号化することができる。この取り決めは、バックアップデータファイルの全体の大きさ内で<seekPoint>フィールド416のややきついパッキングを非常に少ないコストで可能にするが、この最適化は本発明に決して重要ではない。各データブロックは、圧縮することができ（関連<dataBlockPtr>419の<packFlag>422によって指示されている）、次に暗号化することができる。各<dataBlock>405に対する暗号化キーは、バックアップデータファイル（例えば、144）内に保持されていない。すなわち、前述のように、キーは、関連ファイルデータブロックを参照するバックアップディレクトリファイル内に暗号化形式で存在する。一般的には、<dataBlock>セクション405は、バックアップデータファイル内で断然最大のセクションである。暗号化処理の一部として、チェックサムは、ブロックそのもの又はそのためのポインタのいずれかの破損に対して迅速なチェックを容易にするために各<dataBlock>405に付加される。

【００４３】<fInfoPtrs>セクションの定義は432で始まる。特に、このセクションは２つの可変長アレイの<indexRange>433及び<fileInfoData>436からなる。前述のように、<indexRange>レコード433の数は、<header>401の<indexRangeCnt>ﾌｨｰﾙﾄﾞ403によって示されている。<indexRangeCnt>値403及び各<indexRange>レコード433（好ましい実施例では８バイト）の大きさを与えられると、第１の<fileInfoData>レコード436のロケーションは容易に引き出される。通常、バックアップデータファイルは、唯一の<indexRange>433（すなわち、単一の隣接値域のファイル索引）を有するが、エージェント108が串刺し演算（下に述べる）を実行後、多数の非隣接値域が単一のファイルに存在することは可能である。各<indexRange>レコード433は２つのフィールド、すなわち、その各々が好ましい実施例では32ビット値である<indexBase>434及び<indexCount>435からなる。<indexBase>値434は、値域内の第１のファイルインデックスを示している。<indexCount>値435は、値域内のファイルインデックス数を示している。全ての<indexRange>レコード433からの<indexCount>値435の合計はファイル内の<fileInfoData>レコード436の数を示している。好ましい実施例では、各<fileInfoData>レコード436に関連するファイルインデックスは、<indexRange>レコード433によって生成されたファイルインデックス値の配列セットから暗黙のうちに逐次割り当てられる。

【００４４】好ましい実施例では、各<fileInfoData>レコード436は、436～440に示されたように、４つの32ビットフィールドからなる固定サイズのものである。特に、<fileInfoPtr>値437は、関連可変長<fileInfo>レコード408を示している。<fileSize>値438は数バイトの関連ファイルの大きさを示している。<dirInfoCRC>値439は、関連ファイルに対するディレクトリエントリの一部について計算されたハッシュ値（好ましい実施例ではCRC）である。すなわち、可変長ディレクトリエントリの代わりにこの固定サイズの値を使用することはユーザ間のファイル照合のための探索を簡単にする。<partialFileCRC>440は、ファイルの第１の部分について計算されたハッシュ値（好ましい実施例ではCRC）である。好ましい実施例では、それは、（たいていの場合、ファイルの全部である）第１のN_p＝256Kバイトまでのファイルを含む。ユーザにわたる照合ファイルを探索するとき、バックアップアプリケーションは、N_pバイトのファイルをメモリにロードし、ハッシュ値（<partialFileCRC>440）を計算し、次に、<fileInfo>レコード408を照合するためにグローバルデータベース（例えば、145）の端から端まで予備探索を実行する。一致が得られるならば、たいていのファイルがN_pよりも小さいので、通常これをさらにチェックする必要もないけれども、より完全な照合が全<fileCRC>フィールド409を使用して検証することができる。この部分ファイル・ハッシュ技術を利用することにより、一般的に、１回目は<fileCRC>409を算出するために、また、２度目は検索不成功でファイル内容をバクアップするために、ファイル全体を一度に読み込む代わりに、メモリに入りきらないほど大きなファイルのシングル・パス検索が可能である。

【００４５】バックアップデータファイル内に含まれる各ファイルに対する１つの可変サイズ<fileInfo>レコード406がある。<fileCRC>値409はファイルの全内容

についてのハッシュである。すなわち、好ましい実施例では、CRCが使用されている。<bitFields>レコード410は、<fileInfo>レコードのいろいろな属性を示すいくつかの少ないビットフィールドを含む。例えば、<refCnt>フィールド411は、好ましい実施例では、どれだけ多くの外部ファイルがファイルの内容を再構成する際に“参照”されるかを示す２ビットフィールドであり、値０（いかなる外部ファイルもない）１（１つの外部ファイル）又は２、をとることができるが、値３は好ましい実施例では許可されていない。この特定の制限は、<dataPtr>フィールド419の符号化を最適化することだけに負わされている。すなわち、実際は、１つ以上の外部ファイルが前のファイルバージョンを参照することは非常にまれであるけれども、何故多くの外部ファイルを参照することができないかのいかなる理由も理論上はない。<refCnf>フィールド411の値は、<fileInfo>レコード408内に含まれる<fileRef>レコード426の数を示す。<refLevel>フィールド412は、好ましい実施例では６ビットフィールドであり、<fileRef>レコード426内に示されたいかなる外部参照ファイルに対して１＋最大<refLevel>値412又は、<refCnt>411がゼロである場合はゼロであると定義されている。したがって、<refLevel>値412はファイル内容のいかなる部分にもアクセスするのに要する“間接的な処置（indirection）”の最大レベルをカウントする。すなわち、この値は、好ましい実施例では、アクセス時間に関する許容制限を復旧時のファイルの内容に設定するためユーザが設定できるパラメータN_L（一般的には、範囲５～10内に）に限定されている。特定の外部ファイルが参照されるならば、<refLevel>値412がN_L値を超えるときは常に、関連ブロックからのデータは参照による組み込みの代わりに複製される。<isGlobal>ビット413は、所与のファイルがグローバルディレクトリデータベース145の中に入れられるべきであるかどうかを示す。すなわち、それは新規ファイル及び更新済ファイルに対して１であり、他の全てのファイルに対して０である。

【００４６】<seekPts>レコード414は、<seekPts>レコード内の<seekPoint>レコード416の数のカウント<seekPtCount>415（好ましい実施例では32ビット）を含む。各<seekPoint>レコード416は、<seekPoint>レコード416に関連する始めの<logicalOffset>値417からなる。この<seekPoint>レコード416には関連データのためのポインタ<dataPtr>418が続く。<seekPoint>アレイ416は、<logicalOffset>値417に基づいた分類順序で保存され、迅速な２進探索がファイル内のいかなる特定の論理オフセットに対しても<seekPoint>416を得るようにする。各<seekPoint>416によって“包含”されたファイルのバイトの数は、その<logicalOffset>値417を続く<seekPoint>416の<logicalOffset>値417から（又は最後の<seekPoint>レコード416に

対する<fileSize>フィールド438から）引くことによって容易に計算される。好ましい実施例では、<seekPoint>416によって包含される最少のバイト数に対するいかなる断固たる制限もないが、外部ファイルの一部が参照されるにつれてこれは減少（又は増加）できるけれども、一般的には、ブロックはかなり大きい（８Ｋバイト以上）。

【００４７】<dataPtr>フィールド418は、２つの形式の中の一つ、すなわち、このバックアップデータファイル内の<dataBlock>405への<dataBlockPtr>419参照又は外部ファイルへの<exterPtr>420参照のいずれかをとることができる。好ましい実施例では、これらの２つのフィールドの各々は、32ビットからなり、419及び420に示されるような<dword>内の単一種類ビットの値によって識別される。<dataPtr>フィールド418が（419で示されるように０である種類ビットによって決定されるような）<dataBlockPtr>419であるならば、<packFlag>ビットは、関連<dataBlock>405が圧縮されるか否かを示し、好ましい実施例では<dataPtr>418の残りの30ビットを含む<blockOffs>フィールド421は、このバックアップデータファイル内の<dataBlock>405を指示する。前述のように、好ましい実施例では各<dataBlock>405は４バイト境界に基づいて始まるので、30ビットは、ファイル内のいかなる<dataBlock>オフセットも示すのに十分である。<dataPtr>フィールド418が（420で示されるような１である種類ビットによって決定されたような）<externPtr>であるならば、<refFileNo>ビット424は、どのファイルが参照されているか（したがって、２つのファイルのみが好ましい実施例では参照することができる）を示し、<refOffs>値423は、この<seekPoint>416の<logicalOffset>値417からの符号付き相対論理オフセットであり、この<seekPoint>416に関連するデータが得ることができる外部参照ファイル内の絶対論理オフセットを示す。この論理オフセットが与えられたこのような外部ブロックをアクセスすることは、さらに他の外部ファイルを順に参照できる他のバックアップデータファイル内の参照ファイルの<fileInfo>セクション408及び<seekPoint>レコード416を構文解析する必要があることに注目すべきである。すなわち、したがって、参照レベル数に関する制限N_Lがある。好ましい実施例では、<relOffs>フィールド430は、実際にはいかなる制限でもないが、わずか30ビットにすぎないので、極端に大きいファイルを取り扱うとき、この制限は<dataPtr>フィールド418の大きさを拡大することによって容易に取り除くことができるけれども、参照外部ブロックは、所与の<logicalOffset>417の+/-512Mバイトの範囲内で始まらなければならない。

【００４８】任意の<fileRef>レコード426は、どの外部ファイルが<seekPoint>アレイ416の<externPtr>フィールド420によって参照されるかを示す。これらの

＜fileRef＞レコード426は、このファイルに対する＜dataBlock＞レコード405のために使用されるのと同一の暗号化キーによって暗号化される。＜fileRef＞レコード426の＜fileID＞レコードは、バックアップディレクトリファイルに使用される＜fileID＞レコード214とフォーマットが同一であり、参照されている特定のファイルを識別する＜fileIndex＞215フィールド及び＜userIndex＞216フィールドを含んでいる。好ましい実施例では、64ビットからなる＜decryptKey＞レコード427は、参照ファイルのために使用される専用暗号化キーを含んでいる。このキーは、当該ファイルを指し示す<fileID>レコードを保持したバックアップ・ディレクトリ・ファイルに含まれるが、このキーはこの当該ファイルに複製されている。さもないと、該ユーザの個人暗号キーで暗号化した他人のバックアップ・ディレクトリ・ファイルでしか得られない。したがって、キーはここに含まれるけれども、参照ファイルの機密を弱めないために、後述されるようなこのファイルを合法的にアクセスするこれらのユーザだけにアクセスを制限するように符号化される。

【００４９】図８は、仮想ファイルＸに対する＜seekPts＞レコード414の詳細な例を示す。このレコードの＜seekPtCount＞は、450で示されるように５である。したがって、その各々は分解されてその＜logicalOffset＞フィールド（例えば、456）及びその＜dataPtr＞フィールド（例えば、457、458、459）になる５つの＜seekPoint＞レコード451～455がある。第１の＜seekPoint＞レコード451は、456で示されるように０の開始論理オフセットを有し、第２の＜seekPoint＞レコード452は論理オフセット8192で始まるので、この＜seekPoint＞レコードはファイルＸの第１の8192バイト（0-8191）を包含する。第１の＜seekPoint＞レコード451に関連したこれらの8192バイトは、458で種類ビット0で示されるように、451の＜dataPtr＞レコードを＜dataBlockPtr＞と認定するこのバックアップデータファイル内の＜dataBlock＞で得られる。第１の＜seekPoint＞レコード451の＜blockOffs＞フィールド457は、関連の＜dataBlock＞がこのバックアップデータファイル内のオフセット512（すなわち、4×128）で得られるべきあることを示す値128を含み、＜packFlag＞フィールド459内の１ビットは、この＜dataBlock＞が圧縮されていることを示す。同様に、第２の＜seekPoint＞レコード452はファイルＸのバイト8192～11999を包含するが、この＜fileInfo＞レコード内の第１の＜fileRef＞レコード（参照ファイル＃0）に示される外部ファイルの論理オフセット8492で始まるこれらの3808バイトが得られることができる。オフセット8492は、第２の＜seekPoint＞レコード452の＜logicalOffset＞値（すなわち、8192）を＜dataPtr＞内の１の種類ビットで示されるような＜externPtr＞である第２の＜seekPoint＞レコード452の＜dataPtr＞レコードの＜relOffs＞値に付加することによって計算される。すなわち、452の＜refFileNo＞フィールドは、どの＜refFile＞が参照されているかを示す（この場合、0）。第３の＜seekPoint＞レコード453は、ファイルＸのバイト12000～16383を包含し、このバックアップデータファイルのオフセット4080で始まる未圧縮データブロックを示す。第４の＜seekPoint＞レコード454はファイルＸのバイト16384～23008を包含し、これらの6625バイトは、論理オフセット15384（＜logicalOffset＞+＜reOffs＞=16384-1000=15384）で参照ファイル＃１内で得られることができる。すなわち、＜reOffs＞はこの場合負の数であることに注目せよ。第５（最後）の＜seekPoint＞レコード455は、ファイルＸの残りのバイト全てを包含する。例えば、＜fileSize＞が30000であるならば、このブロックは、6991バイト(23009～29999)からなる。これらのバイトは、このバックアップファイルのオフセット8472で圧縮された＜dataBlock＞内で得られる。この例は、＜seekPts＞構造を解釈することはいかに簡単であるかを示し、＜logicalOffset＞フィールドに関する２進探索は、ファイルのいかなる部分をも非常に迅速に突き止めるために使用することができる。

【００５０】＜fileInfo＞レコード408の＜fprnts＞セクション428は、ファイル内容の固定サイズ部分（“チャンク”）について計算されたハッシュ関数又は“フィンガプリント”を含む。これらのフィンガプリントの目的は、前のファイルバージョンの内容を完全に抽出する必要なしにファイルバージョン間の照合チャンクの有効な確率的探索を可能にすることである。このようにフィンガプリント機能を使用するためのアイデアは、カープ＆ラビン[(Karp,Richard M.,and Michael O Rabin)著「有効ランダムパターンマッチングアルゴリズム(Efficient Randomized Pattern-Maching Algorithms)」、Harvard University Center for Research in Computing Technology、TR-31-81、1981年12月]によって最初に着想された。フィンガプリントが、後の章で述べられているように、バックアップデータファイルが遠隔サイトにあるとき、低速度通信リンクを通じて修正ファイルのバックアップを実行するときに特に有効である。好ましい実施例では、フィンガプリントを使用することが絶対必要でないけれども（前のファイル内容はチャンクマッチングに対して明確に生成することができるので）、フィンガプリントは、このような帯域幅最適化を容易にするためにとにかくバックアップデータファイル内に記憶されている。特に、ローカルエリアネットワークを通じてさえも、小さい変更だけを有する大きなファイルをバックアップするときネットワークトラフィックを最少化することは望ましいこともある。ファイル毎に変えることができるフィンガプリントを付けるために使用されるチャンクのサイズは、＜fpChunkSize＞429によって指示され、一般的には、256～8192バイトの範囲内である。＜fpChunkSize＞429に対する0は、いかなるフィンガプリン

31

トも記憶されていないことを示す。好ましい実施例では、＜decrptKey＞レコード427と同様に、＜fprints＞レコード428は、バックアップディレクトリファイルの“fileDecryptKey”セクションからの関連暗号化キーを使用して暗号化される。

【００５１】カープ&ラビンによって詳述されたようなフィンガプリントの基本アイデアは、かなりの量のデータを“スライド”させることが容易であるハッシュ関数を選択することである。換言すれば、チャンク開始ロケーションがファイル内の一つの位置から次の位置に移動されるとき、“最も古い”バイトはチャンク“ウィンドウ”を出て、中間バイトは１つの位置に関してシフトし、新しいバイトはウィンドウに入る。カープ&ラビンは、所与の現フィンガプリントと最も古いバイト及び最も新しいバイトを更新するのが容易であるいくつかの種類の線形のフィンガプリント関数を記載している。例えば、モジュロ256和は特に簡単な事例であるが（簡単すぎて実際有用でない）、CRC及び他の同様な関数はほとんど受容できる。前のファイル内容のチャンクに対するフィンガプリントセットが与えられると、現ファイル内容のチャンクをスライドさせ、各バイトロケーションでの前のファイルのフィンガプリントのいずれかとの照合をチェックすることによってフィンガプリント関数は計算される。一致が得られると、現ファイル内のそのチャンクは、前のファイル内のフィンガプリントと関連するチャンクと一致すると仮定される。フィンガプリント関数は、十分大きくなるように（好ましい実施例では72ビット）選択することができるので、誤った一致の確率（例えば、2^{-72}、又はおおよそ10^{-22}）は、さらなるバリデーションが必要がないように記憶媒体欠陥の確率（一般的には、10^{-15}）よりも小さい。交互に、このスライド式フィンガプリント機構は、ありそうな照合の領域を識別し、次に、古いファイルの内容を抽出し、全比較を実行することによって十分に領域を検証するための探索技術として単独で使用することができる。スライドしないようにだけフィンガプリントを使用することも可能である。この方式は、レコードを移動させない傾向が＊

32

＊ある非常に大きなファイル（例えば、データベース）には特にうまく機能するが、帯域幅消費がたいした問題でないより小さいファイルの場合、全比較はこの場合、実行することができる。他の実施例では、グローバルデータベースは、全ファイルの代わりにチャンクのフィンガプリントで形成することができ、ユーザにわたるファイルの一部の一致を可能にするが、このような方式からの記憶領域における期待値は、余分のオーバーヘッドが必要とされるだけの価値があるようには思えない。

【００５２】好ましい実施例では、各＜fingerprint＞レコード430は、12バイトの全部に対して、フィンガプリント関数（CRC）の９バイト（72ビット）＋関連チャンクの最初の３バイトからなる。これらの余分な３バイトを使用することによって、フィンガプリントが計算され、計算をかなり高速にするバイト毎に基づく代わりにスライドするdword毎に基づいて比較される。しかしながら、速度以外、ネット結果は、バイト毎のスライドのフィンガプリント比較と同じである。ファイルのチャンク当たり１つの＜fingerprint＞レコード430がある。すなわち、しかしながら、ディスク領域を節約するために、いかなる＜fingerprint＞レコード430も、これらのチャンクのためのフィンガプリントは参照ファイルのための＜fileInfo＞408内に既に含まれるので、同一の＜fpChunkSize＞値429とともに外部ファイル参照に完全に含まれ（＜externPtr＞レコード420を介して）、かつ参照ファイル内のチャンク境界上にあるチャンクのためには含まれない。

【００５３】好ましい実施例のバックアップデータファイル内のレコードの特定の配置に多数の可能な変更がある。例えば、他の実施例では、各＜fileInfo＞レコード408は、別々のセクション内にある代わりにファイルに関連する＜dataBlock＞レコード405のセットの直後に置くことができる。すなわち、例えば、この変化は、下記を読み出すために単に定義400を変えることによって表すことができる。

【００５４】

＜bkupDataFile＞ :=＜header＞[＜dataBlock＞*＜fileInfo＞]*

＜fInfoPtrs＞同様に、＜fileCRC＞409及び＜isGlobal＞413のようないくつかのフィールドは、＜fileInfo＞408から＜fileInfoPtrs＞432にまたその逆に移動できる。いくつかのファイルシステム（例えば、ウィンドウズNT、NTFS）では、64ビットファイルポインタは、好ましい実施例の32ビットポインタの代わりに使用される。より多数の参照ファイル又はより多数の参照レベルを考慮するためにフォーマットをわずかに修正することも簡単である。このような変化は基本アイデアに影響を及ぼさなく、ここに記載した好ましい実施例の特定のレコードフォーマットは、本発明の範囲を限定する意図ではない。

【００５５】１．３　グローバルディレクトリデータベースファイル

バックアップデータ及びディレクトリファイルに含まれる情報及びいかに情報がファイルデータ内容を表すために使用されるかがわかれば、照合ファイルを探索するための技術は容易に説明することができる。グローバルデータベースを設計する際に、データベースに入力される数百万（又は数千万）の新規／更新済ファイルがあると仮定される。例えば、スタック（本発明の譲受人）で90のユーザワークステーションを調査すると、全てのディスクにわたる約250,000のユニークなファイル全体が明らかされ、好ましい実施例は、少なくともその多くのバ

33

ックアップノードを有するシステムを処理するように設計されている。したがって、データベース設計に最大の注意が払わなければ、ファイルデータトラフィックを容易に小さくすることができる探索処理によって消費されるネットワーク帯域幅を最少にすることは重要なことである。特に、いくつかの従来のデータベース方式（例えば、Ｂツリー）は、この重要性を考慮に入れて考察され、拒否される。うまく機能する他の種類のデータベースアーキテクチャがあるかも知れないが、好ましい実施例のデータベースの構造は、ここで必要とされる探索の種類に特に有効である。

【００５６】バックアップ処理中、各ノードはグローバルデータベースに対して探索される必要がある何千の新規／更新済ファイルを有することができる。通常、初期バックアップが取られておりさえすれば、そのようなファイルは非常に少なくなるが、初期バックアップが取られない最悪のケースを考えておかなければならない。一方では、既にデータベースに入力された何百万のファイルがある。したがって、はじめに、クライアントが新規／更新済ファイルのその（比較的小さい）リストをサーバに送信し、大きなグローバルデータベースに対して順に探索することを行うバックアップサーバを有するクライアント／サーバ実施例は、共有ファイルシステムよりもネットワーク帯域幅使用において非常に有利である。好ましい実施例の共有ファイル環境の中で探索を実行するためのオーバーヘッドは、この欠点が実際には重要でない点にまで最適化される。

【００５７】ユーザにわたってファイルを照合するために探索する際、通常、照合ファイルサイズ、ファイル名、時間／データ、ファイル内容について計算されたハッシュ値（例えば、CRC）を有すれば十分であると思われている。この方式は間違った一致の有限（わずかであるけれども）の確率を含むが、エラー確率はほとんど全ての実用的アプリケーションに対して許容できる程度に小さい。任意のユーザ呼び出し“完全比較”モードでは、この確率的種類の照合は２つのファイルの内容の完全なバイト毎の比較を始めるのにだけ役立つている。しかしながら、このモードのオーバーヘッドは、それによって得られた確実なレベルにおける実際に無視できるほどの改良を特に考慮して十分大きいので、このような“懐疑的な”反応を呼び出すことは、いやしくもたまに行うのが最もよい。他の実施例では、照合基準は、照合ファイル名又は時間／日付を必要としないようにさらに緩めることができる。たとえば他の全てのパラメータが等しいならば、２つのファイル′REPORT.DOC′及び′REPORT.BAK′は一致であると判定することができる。このテーマには多数の変更例がある。すなわち、例えば、拡張子を除くファイル名だけが比較されるか、又はファイル名の最初の少数（例えば、４～６）の文字だけが、′REPORT′～′REPORT1′のようなわずかなファイル名の

34

変更を含むように比較される。しかしながら、一般的には、ファイルサイズ（又は少なくともある数の最下位ビットのサイズ）及びファイル内容に関するハッシュ値は、バックアップされている新規／更新済ファイルと同一であると既に判定されたデータベース内のファイルのための順序と等しいことが要求される。好ましい実施例では、グローバルデータベースエントリをフォーマット化する際のファイル名（又は他のディレクトリ属性）の可変長の“問題”を処理するために、関連ディレクトリエントリ情報（例えば、ファイル名、時間、日付、及びサイズ）についての32ビットハッシュ（実際は、CRC、<dirInfoCRC>439）は、全ディレクトリエントリの代わりに比較のために使用されている。さらに、ファイル内容についての全ハッシュ値（好ましい実施例では、32ビットCRC）は、ファイルサイズの最下位16ビットと同様に比較される。これらの値の全部が一致するならば、バックアップされているファイルはデータベース内のファイルに一致があるとみなされ、2^{-80}（10^{-24}）よりも小さい間違った一致確率を生じる。明かに、必要とされる一致量は、いかなる所与の環境に対しても受容できる（例えば、CRCのサイズを増加することによって）特定エラー確率のために調整することができ、このような変化は本発明の範囲内になお収まる。

【００５８】グローバルデータベース内のユーザにわたるいろいろな照合ファイルのレベルは、同一ユーザの前のバックアップからの未変更ファイルを識別するために使用される努力レベルよりも一般に全てさらに厳密である点に注目することは有用である。好ましい実施例では、バックアップアプリケーションに全く共通であるように、あるファイルのファイルサイズ、時間、日付、及び名前が前のバックアップから修正されていないならば、デフォルトふるまいはそのファイルが変更されていないとみなすことである。前述のように、ユーザのオプションで、外見上は未変更ファイルの内容の完全な比較又はファイル内容についてのCRC比較のいずれかを実行することは常に可能であるが、確実性のレベルの向上が余分な努力及びオーバーヘッドにみあう価値があるとみなされることはまれである。

【００５９】バックアップされる特定のファイルに対して<dirInfoCRC>439値、<fileSize>438値及び<fileCRC>409値を与えると、好ましい実施例のグローバルデータベースの端から端までの探索は、一致エントリを得ようとする試みで実行される。前述のように、<partialFileCRC>440値は、たいていの場合、全ファイルを含むので、最適化としての全<fileCRC>409の代わりに実際最初に使用される。すなわち、大きなファイルに対しては、したがって、<fileCRC>409は、そのファイルのための<fileInfo>408を含むバックアップデータファイル注意深く調べることによって好ましい実施例では調べられる。各グローバルデータベースエントリは、<di

35

rInfoCRC＞439（４バイト）、＜fileSize＞438（実際、好ましい実施例では、最下位16ビットだけ）、そのファイルのための＜fileInfoData＞レコード436を含むバックアップデータファイルから取り出される関連ファイルのための＜partialFileCRC＞440（４バイト）値を含んでいる。さらに、各エントリは、実際のファイルデータ内容を見つけるために使用することができる＜fileID＞レコード214（６バイト）を含んでいる。したがって、未圧縮データベースに対する全サイズは好ましい実施例では16バイトで固定されている。データベース内にN＝1，000，000のファイルがある場合、探索を実行するためにバックアップ記憶手段101からの全グローバルデータベースをダウンロードすることは、いかなる努力もこのオーバーヘッドを最少にするようになされるならば、16MBのデータをダウンロードすることを必要とする。この量は、全ディレクトリエントリ（例えば、全ファイル名に関して）のデータベース全部のダウンロードに要する量よりもかなり少ないが、ネットワーク上の何ダース又は何百のノードがバックアップを実行できる環境に対してなお極端に大きい。他の実施例では、＜fileSize＞フィールド、＜fileCRC＞フィールド、及び他のフィールド全部は、各グローバルデータベースエントリ内に記憶することもでき、グローバルデータベースファイル145のサイズにおいて少ない費用で、間違った一致の確率及び探索時間の両方をわずかに減少させるが、このような改良は実際の条件ではせいぜいわずかである。

【００６０】グローバルディレクトリデータベース145に関連するデータ転送オーバーヘッド及び探索時間を最少化するために、それは、図９に示されるような２つのレベルに構成され、＜dirInfoCRC＞439及び＜partialFileCRC＞440で使用されているように、CRC機能の特性による探索値の有効ランダム化を利用している。実際に２つの構造502及び505で表されている第１のレベル500の各エントリは、＜dirInfoCRC＞439フィールド及び＜partialFileCRC＞440フィールドのビットの部分集合だけを含んでいる。第２のレベル501の各エントリは、全グローバルデータベースエントリ（例えば、508、509、510及び511）を構成するのに必要な残りのビットを含んでいて、各エントリは破損検査を考慮するために第２のレベルエントリについての16ビットCRCも含んでいる。実際の数は、データベースエントリの数が増えるにつれて一般に増加するけれども、第１のレベル500で含まれるビットの数は各グローバルデータベースファイル内で固定されている。第１のレベル500は、バックアッププログラムによってバックアップ記憶手段101からノードにダウンロードされ、その内容（502及び505）は、非常に高い一致の確率を有するエントリのみに（非常に大きい）第２のレベル501への照会を制限するための高速フィルタとして使用されている。好ましい実施例では、ダウンロード時間を最少化するために、第１のレベル500

36

の全エントリは、ビットレベルでパックされ、ダウンロード後にパックを解除されるのに対して、第２のレベルのエントリは簡単にするためバイト整列されている。

【００６１】両方レベルのエントリは＜dirInfoCRC＞439の値によって分類された同一順序で記憶されるので、第１のレベル500のエントリ（例えば、512）のインデックスを与えると、第２のレベル501の対応するエントリ（例えば、513）の位置が容易に計算することができる。換言すれば、第１のレベル500の第ｋ番目のエントリは第２レベル501の第ｋ番目のエントリに直接対応する。第１のレベルエントリは、実際は、カウントアレイ502及び部分エントリアレイ505を使用して、ダウンロード時間を節約するために圧縮された形式で記憶されている。この簡単な圧縮は、エントリが＜dirInfoCRC＞439の値によってソートされているので、連続＜dirInfoCRC＞値439の主な（最上位）ビットが等しいものである傾向があることに注目することによって達成される。したがって、各エントリに対するこれらの主要なビットを記憶する代わりに、$M=2^{m_0}$のエントリのカウント数アレイテーブルが含まれ、ここで、m_0の値は後述されるようなエージェント108によって選択される。第ｊ番目のアレイエントリn_jは、504で示されるようなｊの値を有する主要なm_0ビットを有する連続＜dirInfoCRC＞エントリ439の数を含む。例えば、図９では、n_0は４で、＜dirInfoCRC＞エントリ439の主要なm_0が0であるテーブル505及び501内に最初の４つのエントリを含み、同様に、n_1は３で、整数として解釈される＜dirInfoCRC＞エントリ439の主要なm_0ビットが１である次の３つのデータベースエントリを含む。データベースが作成されると、エージェント108は、ファイル内の全数のグローバルディレクトリエントリ（N）に基づいてm_0の値を選択する。すなわち、典型的な値は64Kよりも大きいNに対して$m_1=16$である。$N=\Sigma n_j$で、ここで和はj=0..M-1の全ての値に関するものであることに注目せよ。ファイル内の＜dirInfoCRC＞439の値が有効的にランダムに分布されているので、n_jの値は、平均N/M及びかなり小さい範囲を有する分布を有する。このように、好ましい実施例ではさらに記憶領域を最少化するために、実際の値n_jを記憶する代わりに、これらの値はカウント数アレイ502内でn_j-n_{min}で示され、ここで、n_{min}は全てのn_j値に対する最小値である。したがって、各カウント数は、ｓ＝シーリング（$log_2(1+n_{max}-n_{min})$）ビットで表され、ここで、n_{max}は全n_j値に対する最大値である。グローバルデータベースファイルが作成され、グローバルデータベースファイルのヘッダに記憶されるとき、値n_{min}及びｓはエージェント108によって計算される。他の実施例では、ハフマンコード又は算術コードをさらにいっそう使用してカウント数アレイ502の大きさを減少することは可能であることもあるが、カウント数アレイ502は第１のレベル500のサイズのわずかな

37

部分だけを構成しているために、このような利益はわずかである。

【００６２】具体的な例は、この簡単な符号化を明かにするための最も容易な方法である。我々がＮ＝1,000,000のデータベースエントリの全部を持っていると仮定する。我々がm_0＝16を選択するならば、Ｍ＝64Kであり、カウント数アレイ内502の平均値は、N/M≒16である。次に、我々がn_{min}＝２及びn_{max}＝30を得ると仮定する。したがって、ｓ＝５ビットであるので、各カウント数エントリn_jは40Kバイトの全部に対して値n_j-2によって５（パック済）ビットで表される（各々５バイトで64Kエントリ）。カウント数アレイを使用しないで、502における各データベースエントリは、ほぼ２メガバイトの全部（1935バイト）に対して<dirInfoCRC>値439の全てのm_0＝16の主要なビットを含んでいるので、この場合にカウント数アレイを使用することは、第１のレベル500のサイズのほぼ1913Kバイトの全部を節約する。節約量は、ｓの値にあまり依存しないことに注目せよ。すなわち、ランダム分布に関するシュミレーションを使用して、m_0＝16であるならば、データベース内の64,000,000のファイルに相当する1024と同じくらいのN/Mに対してさえ、全カウント数アレイ分布の99.9%を十分超える値がｓ＝８ビット又はそれ以下で表すことができる。実際は、節約量が、最少値を生じるm_0値に近い限り、経験的にm_0の選択にあまり敏感でないように見えるが、エージェント108はm_0のいろいろな値に対し第１のレベル500のサイズを最小化することを試みる。すなわち、換言すれば、単にm_0＝16を使用することは、関心のあるたいていの場合にはかなり十分に作用する。

【００６３】<dirInfoCRC>値439の第１のm_0ビットを非常に有効的に表わすために使用されるカウント数アレイに関しては、第１のレベル500の残りはビットレベルでパックされたＮ個のエントリのアレイ505からなる。各エントリは、<dirInfoCRC>値439のｘビット（最上位m_0ビットを超える）及び<partialFileCRC>440のｙビットからなる。値ｘ及びｙは、グローバルデータベース145内のm_0及びエントリＮの全数に基づいてエージェント108によって選択される（そして、グローバルデータベースファイルヘッダに記憶される）。全部の第１のレベル500はダウンロードされるので、アイデアは、一致を確証するために第２のレベル502に対して必要なアクセス数を最少にするため、アレイ505のサイズをトレードオフすることにある。例えば、上記の例からN及びMを使用して、我々がｘ＝10ビット及びｙ＝０ビットを選択するならば、テーブル505は約1220Kバイトの全部からなる（10ビットのそれぞれで1,000,000エントリ）。すなわち、全データベースの全ダウンロードに必要とされる16Mバイトと対照的に、第１のレベル500の全部は約1260Kバイトからなる。我々は第１のレベル500によって示された<dirInfoCRC>値439のm_0＋ｘ＝26ビットを有

38

するので、第１のレベルにおける一致に基づいた間違った第２のレベルの一致の平均確率p_fは、データベース内の<dirInfoCRC>値439のランダム分布と仮定すると、ほぼ$p_f=N/2^{26}$≒1/64によって示される。換言すると、第１のレベルでデータベースをフィルタリングすると、第１のレベルで一致する64の照会の中の約63は、この例でも第２のレベルでの一致を生じる。第２のレベル501の中へのあらゆる照会はグローバルデータベースエントリの残りのフィールドを含むエントリ（例えば、513）の中へのディスクアクセスを含むので、疑似アクセスを最少にすることが重要である。一般的には、範囲1/16～1/256の範囲内のp_fの値は、探索性能とダウンロードサイズとの間の妥当なトレードオフを示す。例えば、この例ではｘを11ビットに増加するならば、第１のレベルのサイズで約125Kバイトのコストでp_f≒1/128に減少する。この例ではｙ=0であるけれども、<partialFileCRC>440のｙビットは、Ｎが非常に大きくなるにつれて又は同一の名前／時間／日付／サイズを有する多数のファイル（すなわち、<dirInfoCRC>439）が異なるファイル内容（及び例えば、<partialFileCRC>値440）とともに存在する異例な場合、トレードオフの範囲を拡張するために使用することができる。エージェント108は、データベース内のエントリの統計量に基づいたデータベース作成時間にこれら全てのパラメータを決定する。好ましい実施例では、m_0＋ｘは常に少なくとも16であり、第１のレベルエントリは<dirInfoCRC>値439の少なくとも16の最上位ビットを含むことを意味するので、<dirInfoCRC>439の最下位16ビットだけが508で第２のレベル501に保持される必要がある。

【００６４】バックアップ処理の初期段階に、好ましい実施例のバックアップソフトウェアは、グローバルディレクトリデータベースファイル145の第１のレベルを、バックアップ記憶手段101から、又はネットワーク帯域幅消費を最少にするためにノードに対するディスク上のディレクトリ内のキャッシュコピーからメモリにロードする。バックアップされる各新規／更新済ファイルに関しては、一致があるかどうかを調べるために第１のレベルのデータベースエントリの探索が実行される。いかなる一致もこのレベルで得られない場合（“いかなる一致するファイルもない”場合）、データベース内のどこにもいかなる一致するファイルもないので、バックアップは、更新済ファイルの場合には前のファイルバージョンからの差を計算することを含むことができるバックアップデータファイルにファイル内容のコピーを始める。一致が第１のレベルで得られるならば、対応する第２のレベルのエントリは検索され、比較される。すなわち、いかなる一致もここで得られないならば、丁度論議された“いかなる一致もない”場合でのようにバックアップを始める。第２のレベルエントリのその通常の位置は関連する第１のレベルエントリの通常の位置と同じなので、

39

対応する第２のレベルエントリの位置は前述のように容易に決定される。一致が第２のレベルで得られる場合、ファイルに関連する<fileInfo>レコード及び<fileInfoData>レコード408、436を含むバックアップデータファイルをさらに照会することは、ファイルのサイズ（例えば、大きなファイルは<fileCRC>409を必要とするであろう）に応じて、及びユーザが“完全な比較”モードを作動させるかどうかに応じていくつかの場合に必要であることもあるが、大部分の場合、第２のレベルでのグローバルディレクトリエントリが一致すればファイル一致を指示するのに十分である。完全に一致したと最終的に判断されれば、このバックアップのためのバックアップディレクトリファイルの<fileEntry>レコード207に含まれている<fileID>214は、グローバルデータベースの中の照合ファイルを指示するために設定されるので、いかなるファイルデータもこのバックアップのためのバックアップデータファイルの中に保存される必要がなく、割り当てられたいかなる新しい<fileIndex>215も、また付加された<fileInfo>セクション408もない。

【００６５】好ましい実施例で使用された特定の第１のレベル探索機構は非常に簡単で、使用することができる多数の他の周知探索技術がある。ここでの主要点は、第１のレベルデータは、ダウンロードされた後、ノードでローカルに全て使用可能であるので、第１のレベル一致を識別するために遠隔バックアップ記憶手段101へのアクセス要求は決してないということである。好ましい実施例では、第１のレベルデータ全部が探索処理中、主メモリにフィットすることができると仮定される。すなわち、これがこの場合でないならば、仮想化（ディスク方式）探索は、周知のアルゴリズムを使用して、かなりゆっくりではあるが、なお同一結果を達成するよう設計することができる。好ましい実施例は、図１０に示されるように、メモリの中に２つのアレイを形成する。主アレイ526は、Ｎ個のエントリを有し、各々は、グローバルデータベースファイルの第１のレベルと同一順序で分類された第１のレベルのエントリから<partialFileCRC>440の<dirInfoCRC>439及びｙビット（528）のｘ＋m_0ビット（527）を含んでいる。換言すれば、アレイ526は、有効的には505の内容のメモリイメージである。ポインタアレイ520は、$T=2^{m_1}$エントリからなり、ここで、m_1は、グローバルデータベースエントリの全数Ｎ及び探索処理を最適化するために使用可能であるメモリ量に基づいて選択されたビット数である。ポインタアレイ520内の各エントリは主アレイ526の中にあるポインタP_kを含んでいる。ポインタアレイ520の中のインデックスは、当のファイルのための<dirInfoCRC>値439の最上位m_1ビットを抽出することによって計算される。例えば、$P_0$521が526の第１のエントリを指示するのに対して、$P_1$522は、この例では当<dirInfoCRC>439のm_1

40

ビットが整数値１を有する第１のエントリである、526内の第５のエントリを指示する。同様に、P_k523は、この例では当<dirInfoCRC>439のm_1ビットが整数値ｋを有する526内の第１のエントリを指示する。各インデックスｋに対して探索される526内のエントリのカウントは連続ポインタエントリ$P_{k+1}-P_k$間の差から容易に得られる。主アレイの終わりの丁度先を指示する余分“ダミー”エントリP_T525は、同一カウント計算がいかなる特別の場合の論理も必要としないで、最後のエントリP_{T-1}524に対して実行されるよう、好ましい実施例ではポインタアレイ520の終わりに付加される。

【００６６】好ましい実施例では、グローバルディレクトリデータベースファイル145を付加されるエントリは、＼BACKUP＼USERパス（例えば、125）から＼BACKUP＼SYSTEMパス（例えば、129）までの移動の一部としてエージェント処理108によってバックアップデータファイル（例えば、144）から抽出される。エージェント108は、いかなる破損エントリもグローバルディレクトリデータベースに付加されないことを保証するために、バックアップデータファイル内に<fileInfoData>エントリ436を含むCRCを最初に検証する。次に、新しいエントリと古いエントリの合併からなる新しいグローバルデータベースファイル145が作成される。好ましい実施例では、バックアップ処理が現データベースファイルを使用し続けるよう、新データベースは、暫定名の下でエージェント108によって最初に作成される。一旦新しいファイルが完了すると、その名前は、次にその後のバックアップ動作によってアクセスされる有効グローバルディレクトリデータベースファイル名に変更される。好ましい実施例では、グローバルディレクトリデータベースファイルの名前は、形式GDnnnnnn.GDDを有し、ここで、nnnnnnは、新しいグローバルディレクトリデータベースファイルが付加される度に増分される数である。例えば、第１のファイルはGD000001.GDDであり、第２のファイルはGD000002.GDD、等である。このようなファイルの小数（一般的には１～４）の最も最新のバージョンだけが保持されている。すなわち、一旦古いバージョンがもはや使用中でないと、古いバージョンは削除される。このように、例えば、ある時間後に＼BACKUP＼SYSTEM＼GLOBALディレクトリ127に記憶された２つのファイルGD000138.GDD及びGD000139.GDDがあることもある。すなわち、バックアップ動作が開始される度に、バックアップ処理は、使用可能なグローバルディレクトリデータベースファイル145の“最新”バージョンを選択する（この例ではGD000139.GDD）。

【００６７】従来のデータベース設計（例えば、新しいエントリを追加するためにＢツリー構造を使用する）とは対照的に、データベースを完全に再度書き込むこの方法によって、前述の最適化探索構造が保持されることが可能となる。幸にも、バックアップに固有な“バッチ”

41

動作モードは、このような方式をこのアプリケーションで受容できるようにする。しかしながら、一旦バックアップシステムがしばらく使用中であると、各新しいバックアップのためのグローバルデータベースのための付加エントリの数は、特に、新規ファイル及び更新済ファイルだけがデータベースに追加されるので、全データベースサイズの非常にほんの少数になる。例えば、グローバルデータベース内に1，000，000のエントリがあることもあるが、新しいバックアップ処理は、２，３ダースのエントリだけが追加することができる。この場合、エントリグローバルデータベースを再書き込みすることは、極端に遅い処理であることがあり、各バックアップ後に新しいデータベースをダウンロードすることは遅いこともある。好ましい実施例では、このようなオーバーヘッドを最小にするために、エージェント処理108は、＼BACKUP＼SYSTEM＼GLOBALディレクトリ127内の“更新”ディレクトリファイル147に通知できる。基本的には主グローバルデータベース145に構造上同一であるこれらの更新ファイル147は、グローバルデータベースに追加される新規エントリだけを含む。これらの更新ファイルのいくつかは全く小さいこともあるので、エージェント108は、m0＝0、x＝16、及びy＝0の場合の簡略フォーマットでこれらを記憶するように選択でき、それでいかなるカウントテーブル502もない。

【００６８】好ましい実施例では、各更新ディレクトリファイルは、関連“ベース”グローバルデータベースファイルにリンクするファイル名を与えられている。すなわち、ネーミング取り決めはGUxxxnnn.GDUであり、ここで、nnnはベースグローバルデータベースファイル名の最後の３つの数字であり、xxxは更新番号である。例えば、ファイルGU003138.GDUは、ベースファイルGD000138.GDDの第３番目の更新である。２、３のグローバルデータベースファイルだけがいつでも保持されているので、３つの数字nnnは、好ましい実施例では関連グローバルデータベースファイルを明白に識別するのに常に十分である。

【００６９】バックアップソフトウェアは、通常、バックアップ記憶手段101からダウンロードされる最後のグローバル／更新ディレクトリファイルのローカルディスク上に簡単なキャッシュを保持しているので、データベースの第１のレベルのダウンロード処理速度を増すことができる。好ましい実施例では、このような更新ファイルは、その関連主データベースファイルのための全ての更新（すなわち、差分更新）を含むので、バックアップ処理のみは、いつでも最新の更新ファイルをダウンロードしなければならない。他の実施例では、その代わり、この更新は、本来増分であることがあるので、全ての更新ファイルはダウンロードされねばならないし、又は増分更新及び差分更新の両方が記憶でき、バックアップソフトウェア局部キャッシュ論理にもっと多くの最適化適

42

応性を与える。一旦更新リストが所定のサイズ（例えば、グローバルデータベースのサイズの10%）に達するか又は所定数の更新ファイル（例えば、500）が追加されると、エージェント108は、主データベースファイル及び更新データベースファイル内の全てのエントリを含むまったく新規のグローバルデータベースファイル145を再形成する。どのくらい頻繁に新規データベースファイルが形成されるかを左右するこれらの特定の設定は、エージェントノード107上のバックアップシステムアドミニストレータによって制御される。新規グローバルディレクトリデータベースを形成した後さえ、エージェント108は、一時古いグローバルディレクトリデータベースを残したままにしておくことでき、さらに古いグローバルディレクトリデータベースに更新を加え続けることができるので、バックアップソフトウェアのローカルキャッシュ論理はそのダウンロード方式を最適化することができる。一般に、バックアップソフトウェアは、バックアップ記憶手段101から第１のレベル500の全グローバルディレクトリデータベースファイル145をたまにだけダウンロードをする必要があり、したがって、各バックアップ動作ための起動時間を最小化する。

【００７０】１．４．他のバックアップファイル

図３は、前述のファイルタイプ以外のいくつかのファイルタイプを示している。これらのファイルの大部分は、余分であるか（すなわち、他のファイルから再生することができる）又はせいぜい本発明に付属しているかのいずれかである。これらのファイルの内容及び使用の簡単な説明がここで十分行われている。

【００７１】前述のように、インデックス値域ルックアップファイル（例えば、151）は、エージェント処理108によって各ユーザのために形成され、保持される。このファイルは、移動バックアップデータファイル及びバックアップディレクトリファイル（例えば、148，149）の内容から構成される。それは、各バックアップディレクトリ及びバックアップデータファイルのディレクトリ／ファイルインデックス値域をそれぞれ指示するテーブルを含んでいる。したがって、このファイルは完全に余分で、バックアップディレクトリ／データファイルのための内容のテーブルとみなすことができ、その内容は、迅速な２進探索が、このような探索を実行するために順に各ファイルを開かねばならない代わりに、ファイルがユーザのための所与のディレクトリ／ファイルインデックスを含むと同時に決定することを可能にするように構成されている。このファイルは暗号化されない。

【００７２】バックアップログファイル（例えば、150）も余分なファイルで、各ユーザのためにエージェント108によって形成され保持される。このログファイルは、一般には、ユーザに使用可能なバックアップのリストを表示するために復旧時間に使用され、バックアップが生じるとき、ユーザによって提供される注釈文字列を

43

含んでいる。このファイルがない限り、復旧ソフトウェアは、全く遅いことがあるこのようなリストを表示するために多数のバックアップディレクトリファイルを開かなければならない。このファイルの内容は、バックアップディレクトリファイルに適用された同一の暗号化キーを使用して暗号化される。

【００７３】ユーザアカウントデータベースファイル146は、バックアップアドミニストレータソフトウェアによって保持される。それは、全ての許可されたバックアップユーザのためのアカウントレコードを含んでいる。特に、それは、後の章で述べるように、ユーザ名（例えば、JOHN）のリスト、ユーザディレクトリ名（例えば、USER2）、<user ID>値、並びに各ユーザのための暗号化キー及びパスワードキーを含んでいる。このファイル内の各ユーザに関連したレコードの大部分は、ユーザ専用パスワードを使用して暗号化される。

【００７４】パスワードログファイル（例えば、140）は、ユーザパスワードの変更を実行するために使用される。この動作は、下記により詳細に述べられているが、このファイルは、基本的に各ユーザがエージェント８にパスワード変更要求を通知するのを可能にしてくれ、順にユーザアカウントデータベース内のユーザのパスワードキーフィールド及び暗号化キーフィールドを更新し、ユーザのバックアップディレクトリファイル（例えば、148）を再暗号化する（例えば、148）。

【００７５】前のディレクトリファイル（例えば、141）は最後のバックアップ動作からのディレクトリ情報を含んでいる。その内容は、冗長であり、バックアップディレクトリファイル（例えば、148）から再構成することができる。しかしながら、バックアップディレクトリファイルと違って、前のディレクトリファイルは、ユーザパスワードを必要とするキーによって暗号化されない。したがって、ユーザが自分のパスワードをタイプインする必要なしに予め予定された時間（例えば、真夜中）にバックアップ動作を始めることができる。このファイルが消失又は破損された（望ましくはまれな）場合、このファイルは再構成することができるが、ユーザがパスワードを入力した後だけである。

【００７６】ユーザ優先ファイル（例えば、142）は、設定できるパラメータの値（例えば、N_U、N_D）のようなユーザ選択優先、ファイルがバックアップから除外される規定、等を含んでいる。

【００７７】システム内のこれらのファイルの全ては、いかなる商業的に使用可能なテープバックアップパッケージをも使用してテープのために容易にバックアップすることができることに注目すべきである。これらのファイルの大部分の読み出し専用特性のために、ネットワークセキュリティが破られなければ、ユーザによって引き起こされるデータ破損をする機会はほとんどないことに注目せよ。したがって、テープバックアップは、ほとんどすべての場合に悲劇的な故障回復の役割にゆだねられる。

44

【００７８】１．５　遠隔バックアップ

移動（例えば、ノート型パソコン）又は遠隔（例えば、ホームオフィス）ユーザに関しては、バックアップはしばしば非常に問題となる。バックアップ訓練を強化するための典型的な障害は、バックアップ装置を買うか、又は携帯することは通常望ましくないため、及び使用可能である場合、ネットワークとの接続がしばしば非常に低速度（例えば、モデム）であるための両方のために、拡大される。しかし、遠隔コンピュータ上のデータは、いかなるネットワークのノード上のデータと同じくらいにも重要であることもあるので、バックアップは同様に重要である。本発明は、多くの場合にこの問題のためのかなり簡単にであるが非常に有効な解決策を実現できる。

【００７９】基本的なアイディアは、ネットワークのために低速度リンクを通じてバックアップを使用することにあり、ファイル変更だけが送信されるためにファイルバージョン間の差を識別するためのリンクを通じて複写ファイルを<fprints>レコード428に送信するための要求を取り除くための好ましい実施例の複写ファイル識別方法による。一般的には、遠隔コンピュータ（例えば、104）がかなり高速なリンクでネットワーク106に直接接続されている場合、可能であるならば、初期バックアップを実行することは望ましいことである。さもなければ、第１のレベル500のグローバルディレクトリファイル145の初期ダウンロード及び一般的には将来変わらないユーザ固有ファイルの送信は、初期バックアップを全く緩慢にする。しかしながら、高速接続が可能でない場合、初期バックアップは、数時間を要するけれども、なお遠隔で実行することができ、通常、複写ファイルIDからかなり利益を受ける。一般的には、その後のバックアップは、数分足らず遠隔で実行することができる。本明細書の全体にわたって述べられたローカルキャッシング方式は、この場合、性能に明かに重要である。さらに、この機能性は好ましい実施例に必要でないけれども、差分動作をさらに高速にするためにローカルディスク上の前のバックアップから<fprints>セクション428をキャッシュすることは有用である。

【００８０】遠隔復旧動作はローカルアクセスよりも遅いが、若干の小さいファイルを復旧するのに要する時間は一般にはかなり満足できる。たいていの場合、複写ファイルIDは好ましい実施例ではダウンロード時間を減少する際に少しも有用でないために、遠隔低速度リンクを通じての完全復旧は、薦められない。

【００８１】２．機密

前述されているように、重要な（多少微妙でもあるけれども）機密問題は、ユーザにわたる複写ファイルを識別し、参照できるために生じる。簡単な例として、ユーザ＃１は、バックアップデータファイル内に既にセーブさ

45

れていて、グローバルディレクトリデータベースに入力されているファイルＡ、Ｂ、及びＣを有し、ユーザ＃２はファイルＸ、Ｂ、及びＺを有すると仮定する。ユーザ＃２がバックアップを実行するとき、バックアップソフトウェアは、前述の技術を使用して複写ファイル（Ｂ）の存在を検出し、ユーザ＃１のファイルＢを参照する＜fileID＞レコード214をバックアップディレクトリファイルに入力する。これは全て緻密である。すなわち、しかしながら、複写ファイルＢを見つけるために、ユーザ＃２は、たとえユーザ＃１が機密を守りたいファイルであることもあるファイルＡ及びファイルＣであってもユーザ＃１のデータファイルの全てに効率的にアクセスする。たとえバックアップソフトウェアが、この章に述べられた防止対策のいずれでもない場合、これらの非複写ファイルにアクセスする際に直接ユーザ＃２をサポートしないように適切に設計されていると仮定しても、賢いハッカーは、（巧みなリバースエンジニアリングの尽力によって）バックアップデータファイル内のバックアップデータ記憶手段101上にあるユーザ＃１のファイルの全ての内容をアクセスできる。この種のアクセスは、好ましい実施例のように、顧客の専用データが専用のままであることを顧客に再保証することを期待するいかなる製品でも許可されてはならない。機密に対するこの必要性はユーザのワークステーション上に保持されたパーソナルデータにぴったりと関連しているものでない。すなわち、それは、ある種の商取引の話し合い、従業員の給料、他のパーソナルデータ等のような重要な企業情報をもしばしば含んでいる。

【００８２】２．１　専用ファイルの機密の守秘

本発明は、ファイル毎に基づくユーザのデータへのアクセスを制限するために暗号化を使用する簡単で新規な技術を含んでいる。特に、実際ファイルの有効なコピーを持っている（又はかって持っていた）これらのユーザだけがその特定のファイルを参照できる。バックアップセット内の各ファイルのデータは、暗号化キーがファイルのデータそのもののフィンガプリント（例えば、CRC）に基づいている場合、バックアップデータファイル内に暗号化された形式で記憶されている。したがって、これらの“暗号化”フィンガプリントそれ自体は、パスワードを供給することによってのみユーザへアクセスできるキーによってそれ自体暗号化されているバックアップディレクトリファイルの＜fileDecryptKey＞セクション内に記憶されている。さらに、前述のように、バックアップデータファイル内の＜fileRef＞レコード426も参照ファイルのための＜decryptKey＞レコード427を含むが、これらのレコードも、“間接”アクセスを防止するために参照ファイルの暗号化のフィンガプリントを使用して暗号化される。このように、ユーザは、たぶん、自分自身のファイルのコピーについてのフィンガプリントを計算することによって得られる正しい暗号化のフィンガプリントを有するならば、ファイルのデータを首尾よく復号化することだけができる。このように、ユーザは、他のユーザの専用ファイルの暗号化バージョンだけをアクセスするが、同時に、共通であり、（したがって専用ではない）ファイルを復号化することができる。好ましい実施例では、各64ビット暗号化のフィンガプリントは、代数学的に、＜fingerPrint＞値430とは無関係であり、ファイル内容のCRCと最初の256Kバイトについての簡単な非線形チェックサム機能の組み合わせである。この方式の１つの興味ある属性は、ファイルの“元の”所有者が自分のパスワードを忘れた場合、所有者は自分のファイルのアクセスを実際に拒否されるのに対して、他のユーザは共有するファイルへのアクセスを継続することができるというものである。

46

【００８３】ユーザが、所定のファイルの単なる存在又はファイルの名前の機密を守りたいかもしれないことも事実である。したがって、好ましい実施例では、各バックアップディレクトリファイル（例えば、143）の＜volumeDirInfo＞セクション200は、＜fileDecryptKey＞セクションのために使用されるのと同じユーザ特有キーによっても暗号化される。個別のユーザが、ファイルの名前を知らないで、所与のファイルの（暗号化）データ部分へアクセスできるように十分な情報がバックアップデータファイルに記憶されているので、招かざるユーザが自分のパスワードを知らないで他のユーザのディレクトリ／ファイルツリーを“詳細に調べる”ことはできない。

【００８４】２．２　“裏口”データアクセス方式

データ機密に対する必要性は、自分のパスワードを作成することに抵抗があるか又は自分のパスワードを作成できない従業員の場合、その知的所有権（例えば、コンピュータファイル）へのアクセスを保持するために企業環境で会社の権利とバランスさせねばならない。このような場合は、従業員が自分のパスワードを忘れ、不機嫌になり、又は身体障害者になる事故又は致命的な事故の犠牲者である場合、容易に発生することができる。一般的には、ユーザのデータの現バージョンはワークステーションディスクから直接使用可能であるが、企業がユーザのバックアップデータセットをアクセスすることが重要である明かなシナリオがある。

【００８５】バックアップソフトウェアによって、ユーザだけが、バックアップセットデータに対する管理上のいかなる“裏口”もなしに自分のパスワードを設定することを可能にすべきであることが最初に考えられた。しかしながら、このような方式は、前述の“災難”シナリオのいずれかでもそのデータを復旧できる能力を企業に与えないことがはっきりした。さらに、エージェント処理108がこのような環境で実行することが非常に困難になるか又は不可能になるパスワードを変えること及びバックアップセットを合併することのような所定の操作が

47

あることがすぐに明かになる。したがって、好ましい実施例は、ユーザ間で非常に高いセキュリティ及び機密を保持するように設計されているが、バックアップアドミニストレータは、必要ならば、ユーザのバックアップデータをアクセスできる。

【００８６】所定のパーソナルファイルの最高機密を保持することを要求するユーザにとっては、いくつかの選択権がある。ただし、これらの選択のいくつか（又は全て）はユーザのマネージャには受容できないこともある。第一に、ユーザは本発明のバックアップソフトウェアを使用しないようにすることができる。第二に、ユーザは、別々の暗号化ユーティリティを使用して自分のローカルディスク上の当のファイルを暗号化できる。第三に、ユーザはバックアップセットから当のファイルを除外する。好ましい実施例では、アドミニストレータは各ユーザ優先ファイル（例えば、142）にアクセスする。このユーザ優先ファイルは、監査がバックアップされていないこれらのファイル及びディレクトリの管理によって行われることができるように排除／包含リストを含んでいる。

【００８７】２．３　暗号化キープロトコル

暗号化を使用するいかなるシステムでも、暗号化キーがいかに処理されるかに細心の注意が払われなければならなく、本発明はこの規則に対する例外ではない。この章は、機密を保証するために好ましい実施例では暗号化キーの生成及び使用を述べている。

【００８８】図１１に示されるように、新しいユーザアカウントが好ましい実施例のバックアップシステムにアドミニストレータによって追加される場合、アドミニストレータソフトウェアは、ユーザのバックアップディレクトリファイルを暗号化するために使用されるユーザ特有のランダム固有暗号化キー（userDirKey541）を生成する。理解されるように、これらのディレクトリファイルの“fileDecryptKey”セクションは、バックアップデータファイル内でファイルデータ542を暗号化するために使用されたキー543（ファイル内容に関するフィンガプリント機能から生成された）キーを含んでいる。userDirKey541は、ユーザ供給パスワード540により暗号化されているUserAccountDatabaseファイル146の中に入れられる。このパスワード540は、最初にユーザによってアドミニストレータに供給することができるか、又はそれはアドミニストレータによって選択することができ、ユーザに与えることができる（通常、最初の使用の際にこのパスワードを変更するための指示とともに）。

【００８９】好ましい実施例のアドミニストレータソフトウェアは、アドミニストレータパスワードを使用して暗号化されるUserAccountDatabaseファイル146の別々のセクション内にユーザパスワード540及びuserDirKey541も記憶する。実際、記憶されるものは、パスワードから生成される暗号化キー（好ましい実施例ではメッセージ

48

要約）であって、パスワードそのものではない。したがって、アドミニストレータは、必要ならば、ユーザのディレクトリファイルを復号化するための“裏口”経路を有する。さらに、アドミニストレータは、ハッカーがユーザのパスワード540及び／又はuserDirKey541へどうにかしてアクセスする恐れから守るためにuserDirKey値を時々変更し、全てのuserDirファイルを再暗号化するようにエージェント処理108を構成することができる。このような変更は完了するためにいくらかの時間を必要とするけれども、好ましい実施例では、ユーザのためのバックアップディレクトリファイルは、この動作中“オンライン”のままである。これは、実際、UserAccountDatabaseファイル146内の各ユーザのためのアカウントエントリに２つのuserDirKey値（現在値及び“古い”値）を記憶することによって達成される。復号チェックサムが現在のuserDirKey値を使用して失敗するならば、好ましい実施例のバックアップソフトウェアは、その代わりに、古いuserDirKey値を自動的に試行する。したがって、エージェント108は、最初に古いuserDirKey値を現在のuserDirKey値になるように設定し、次に、新しい現在のuserDirKeyを設定し、最後に、全てのバックアップディレクトリファイルの再暗号化を始める。再暗号化処理中いつでも、２つのキーの中の一つが機能する。

【００９０】ユーザは、自分のパスワードログファイル（例えば、140）内のパスワード変更リクエストを通知することによっていつでも自分のパスワードを変更することができる。このリクエストは、現在のuserDirKey値541で暗号化され、新しいパスワードを含む。エージェント108がこのリクエストを処理する余裕ができると、エージェント108は、UserAccountDatabaseファイル146内のユーザのアカウントエントリを新しいパスワードにより再暗号化し、ユーザのパスワードログファイル（例えば、140）を更新することによってリクエストを受け付ける。しばらくは、最新のパスワードのリストがパスワードログファイル内に保持されているために、ユーザは、最新パスワードを使用して暗号化された新しいパスワードを使用することができる。ユーザがuserDirKey541にアクセスする（例えば、復旧を実行する）必要がある場合、ソフトウェアは、UserAccountDatabase146内のuserDirKey541をアクセスするために最新パスワードを使用する。すなわち、故障の際に、パスワード“履歴”はアクセスされ、古いパスワードは、機能するまで自動的に試行する。したがって、ユーザは、数回パスワードを変えることができ、自分の変更リクエストを処理するためにエージェント108を待つ必要なしに操作し続ける。CRCは、パスワードが正しいことを検証するために全ての場合にこれらのファイル内に埋め込まれていることに注目せよ。ユーザが自分のパスワードを失念したり又はリクエストが保留している間、自分のパスワードログファイルを誤って削除するという最悪の場合、アドミ

49

ニストレータは新しいパスワードをユーザに容易に発行できる。

【００９１】好ましい実施例ではアドミニストレータが構成できる選択として、所定のレベルのセキュリティの保証に役立つために、バックアップソフトウェアは、ユーザが周期的方式でパスワードを変えるように促し、全てのパスワードが最少長を有し、（再使用されない）ことをチェックすることができる。他の実施例では、基本的な裏口として、認定された第三者だけが復号化できる公開キーアルゴリズムを使用して暗号化されたファイル内に全てのユーザのパスワード及びuserDirKey値のログをアドミニストレータソフトウェアに保持させることができる。この場合、アドミニストレータがパスワードを復旧する能力を失った場合、第三者は、リクエストの妥当性をチェックするコストをカバーし、このサービスの不まじめな使用をやめさせるためにかかる多分かなりの料金と引き換えにアドミニストレータパスワード及びユーザパスワードを回復することができる。

【００９２】好ましい実施例の一つの目的は、ユーザがパスワードを入力することなしにバックアップを実行することを可能にすることにある。この能力は、ユーザがいないとき、予定済みバックアップを実行する一般的な場合に特に重要である。同時に、データを復旧するためにパスワードを必要とすることが明かに望ましい。幸にも、この機能は次の通りに容易に実施される。各バックアップ中、バックアップソフトウェアは、この目的のためにだけ特別のユーザ特有キー（ｕｓｅｒＰｏｓｔＫｅｙ）を使用して暗号化されたバックアップディレクトリファイル（例えば、１４３）を通知する。userPostKey値は、UserAccountDatabaseファイル146のユーザアカウントエントリ（ユーザパスワード540を使用して暗号化される）の中に含まれる。すなわち、このキーは、パスワードを入力なしに使用可能であるようにローカルワークステーション上に記憶することもできる。＼BACKUP＼SYSTEM経路122へのバックアップディレクトリファイルの移動の一部として、両方のキーにアクセスできるエージェント108は、userDirKey541を使用してこのファイルをその後に再暗号化する。好ましい実施例では、ハッカーは、理論上、ローカルワークステーション及びバックアップディレクトリファイル（例えば、143）からuserPostFileをコピーすることができるので、システムは、バックアップセットの機密を保持するためにネットワークセキュリティ及びローカルワークステーションのセキュリティに依存する場合、エージェント108はバックアップディレクトリファイルを移動するまで、バックアップディレクトリファイルが最初に通知される時から、このように暫時の期間がある。エージェント公開キーを使用して、公開キー暗号化アルゴリズムでディレクトリファイルを通知することによって他の実施例におけるこの制限を抑えることができる。すなわち、しかしながら、

50

一旦ハッカーが（userPostKeyの認められていないコピーを得るために）ユーザのワークステーションにアクセスすると、バックアップデータセットの機密はたぶんみんなの関心があるところではなくなるという事実を考慮すると、このような方式は過剰に見える。

【００９３】さらに、バックアップソフトウェアは、userPostKeyによっても暗号化され、したがって、パスワードなしにアクセスすることができる前のディレクトリファイル（例えば、141）を保持する。このファイルは、最新バックアップのための全てのディレクトリ情報のコピーを含み、次のバックアップで未変更ファイル及び修正済ファイルの識別を可能にする。好ましい実施例のソフトウェアは、ネットワーク帯域幅を最少にするためにローカルワークステーション上にこのファイルのキャッシュされたコピーを保持することもできる。このファイルはファイルデータを暗号化するために使用される暗号化のフィンガプリントを含まないので、前のディレクトリファイルの内容がどういうわけか傷つけられるならば、ディレクトリ情報の知識（ファイルデータ暗号化キーとは対照的に）だけは、最悪の場合に処理される。CRCをチェックすることによって検出することができるこのファイルが破損又は削除されたまれな場合、好ましい実施例のバックアップソフトウェアは、前の（暗号化）バックアップディレクトリファイルから前のディレクトリファイルを再形成する。ただし、このような再形成は、ユーザが自分のパスワードを入力する必要がある。

【００９４】３．復旧処理

好ましい実施例は、復旧されるバックアップセットを選択する２つの主要な方法を実現できる。従来の方法では、ユーザには、各々が、（例えば、150のようなバックアップログファイルからの）バックアップの時間、日付、及び説明で識別され、ユーザが所望のバックアップセットを選択する前のバックアップ動作のリストを表示する。他の方式では、ユーザは現ディスク内容からファイルを選択し、全てのバックアップセットに含まれるそのファイルの全ての前のバージョンのリストを表示される。このリストは、新しいバージョンがバックアップされているときに示しているカレンダー上の選択できるアイコンのセットとして一般的には表示される。このリストの最初の生成の速度を増すために、一旦ユーザがファイルを選択すると、好ましい実施例では、＜lastVersion＞フィールドは、前述のように、各ファイルの全ての独特なバージョンの直接リンクリストを与えるために各＜fileEntry＞レコード207に付加されている。

【００９５】好ましい実施例では、一旦バックアップセットが選択されると、バックアップ記憶手段101からのデータを復旧する２つの方法がある。第１の技術は、基本的に“従来の”復旧動作と同一である。ディレクトリ情報が関連バックアップディレクトリファイルから取り

51

出される場合、ユーザには復旧のために使用可能であるファイルのツリーが表示される。ユーザが所望のファイルに“タグを付け”、復旧受け手を指定した後、復旧ソフトウェアは、バックアップデータファイルからファイル内容を検索し、このファイル内容を受け手に書き込む。

【００９６】第２の復旧パラダイムは、データをアクセスする際に非常に多くの柔軟性を与える。一旦ユーザがバックアップセットを選択すると、ファイルセットは、特別のファイルシステムドライバによって読み出し専用ディスクボリュームとして“取り付けられる”。このドライバは、好ましい実施例ではインストール可能なファイルシステム（IFS）として実施されている。すなわち、他の実施例では、ディスクボリュームは、通常のディスクボリュームのオンディスクフォーマットがバックアップセットの内容に一致するように合成されるブロック装置ドライバを使用して取り付けられる。その根源的な構造にもかかわらず、このドライバは、いかなるアプリケーションもファイルをアクセスするのに必要な全てのオペレーティングシステムに特有な機能を提供する。例えば、ユーザが関連バックアップセット内でバックアップされるスプレッドシートファイルを見たいならば、一旦バックアップセットが取り付けられると、ユーザは自分のスプレッドシートプログラムを簡単に実行でき、最初にファイルをローカルハードディスクにコピーする必要がなく、取り付けられたボリューム上に直接ファイルを開くことができる。すなわち、その代わりに、ユーザは、取り付けられたボリュームから自分自身の好みのファイル管理アプリケーションを使ってユーザのローカルハードディスクにいかなるファイルも簡単にコピーすることができる。この方式は、ユーザが、自分自身のツール及びアプリケーションを使用して、その代わりに、まれに使用されているために精通していない専用復旧アプリケーションによって、より直感的な方法で、自分のバックアップデータをアクセスすることを可能にする。この方式は、従来の復旧プログラムを使用してバックアップセットからより古いバージョンを復旧するとき、ファイルの現バージョンを誤って上書きする共通の問題の周りをも処理する。

【００９７】バックアップ記憶手段101はランダムアクセス装置であるため、<externPtr>参照420を追従するのにわずか数シークを必要することができるけれども、いかなるファイルをもアクセスするのに必要とされる時間は典型的なディスクアクセス時間に匹敵することを確認せよ。関連バックアップディレクトリファイルは、バックアップディレクトリツリー内の任意のところのいかなる特定のファイルへのアクセスも関連<fileInfo>レコード408の中で読み出すこと及びデータブロックをアクセスすることだけを含むバックアップセットが一旦選択されると、非常に高速にディスクからロードされる。

52

このように、好ましい実施例では復旧動作は、テープバックアップシステムからの匹敵する復旧動作よりも殆ど全ての場合ずっと速い。特に、ファイルアクセスは十分速いので、取り付けられたバックアップボリューム上のファイルをアクセスすることは元のディスクドライブ上のファイルをアクセスすることよりも通常かすかに遅い。他の実施例は、実際に全てを書き込むことはローカルディスク上にオーバーフローできるローカル一時キャッシュ内に記憶され、書き込みできるようにさせておくドライブソフトウェア論理を付加することによって取り付けられたバックアップボリュームのこの“実時間”特徴をさらに利用することができる。一旦ボリュームが取り外されると、この一時キャッシュへのいかなる書き込みも止められる。このような方式は、例えば、ユーザがボリュームを取り付け、編集又はデータベース分類のような一時的“適所に更新”動作を実行し、この動作から関連結果を検索し、次に、ボリュームを取り外すことを可能にする。すなわち、実際に、ユーザは、更新動作を実行するためにちょうどいい時に自分のディスクドライブを一時的に取り戻す。

【００９８】好ましい実施例の復旧方法は、初期バックアップ後の各バックアップ動作は実際上“増分”バックアップであるけれども、復旧のために表示された画像はバックアップの時にソースディスク上にある全てのファイルを含み、前述のように、これらのファイルの全ては実時間でアクセスできる点でも、多少独特である。バックアップ記憶手段101のランダムアクセス特徴は、ファイル変更だけを復旧させておき、したがって記憶コストで最大節約を実現できるが、全てのファイルに対する実時間アクセスを依然として考慮している。

【００９９】他の主要な長所は本発明に生じる。一旦システムが設置され、構成されると、バックアップ記憶手段101上に十分自由なディスク領域があることを確認すること以外、いかなるバックアップ又は復旧動作を実行するためにいかなるアドミニストレータインタラクションも必要とされないことに注目せよ。好ましい実施例で達成された高いレベルのデータリダクションによる実際上の事例であるように見える十分なディスク領域を実現できるコストは許容できる程度に低いと仮定すると、バックアップシステムは非常に低い保守コストである。比較すれば、たいていのテープのバックアップシステムは、テープを周期的に変え、所与の復旧リクエストのためにテープを取り付けるためにオペレータの介入を必要とする。すなわち、たとえ（高価な）テープ又は光ディスクジュークボックスハードウェアを有していても、このような動作は、本発明の実時間特徴と対照的にほとんど基本的であるように見える。

【０１００】他の実施例では、本発明は、単一コンピュータに適用することもできることに注目せよ。この場合、バックアップ記憶手段101は、ローカルハードディ

53

スクのセクション又は取り外し可能ディスク装置（例えば、ベルヌーイ、シクエスト）あるいはネットワークディスクボリュームの一部であってもよい。複写ファイルIDの利点はたぶんこの例では重要でなく、述べられた全ての他の重要な長所はやはり適用する。エージェント処理108は、バックアップアドミニストレータとしての役を務めるユーザのために単一コンピュータ上でバックグラウンド処理として実行することができるか、又はエージェントの機能は各バックアップ動作の一部として自動的に実行するように構成することができる。

【０１０１】４．エージェント機能

エージェント処理108は、一般的にはデスクトップPC上のバックグラウンドタスクとしてネットワーク106上のノード107で実行するが、ファイルサーバ100上のソフトウェアタスクとしても実行できる。バックアップアドミニストレータは、後述されるように、完全にはかることができるその位置及び性能の両方においてエージェント処理108を構成する。これらの設定は、バックアップシステムの使用が発展するにつれて時間にわたって変えることができる。例えば、ほんのわずかのユーザだけのためのバックアップシステムでは、エージェント処理108は、アドミニストレータ自身のデスクトップPC上のバックグラウンドタスクとして実行できる。より多くのユーザが追加され、エージェント処理108がより多くの時間を必要とするつれて、アドミニストレータはエージェント処理108を実行するためにネットワーク上のPCを専用にすることができる。結局は、次にネットワークを通じることの代わりにローカルディスクボリュームとしてバックアップ記憶手段101をアクセスできるエージェント処理108を実行することを含むバックアップに専ら専用にされたバックアップファイルサーバを設置することは理解できる。

【０１０２】前述の移動機能及び他のエージェント機能に加えて、エージェントソフトウェアによってアドレス指定しなければならないいくつかの他の問題がある。例えば、バックアップ動作の最中に失敗するバックアップ“クライアント”に関する潜在的な問題がある。同様に、エージェント108そのものも、移動又は串刺し（後述される）演算中に失敗し得る。アプリケーション及びエージェントソフトウェアの両方は、好ましい実施例ではこのような状態を検出及び適切に応動するのに十分活発で、実行される次の時に“それ自身の混乱状態をきれいにする”能力を含んでいる。若干の他のこのような問題が簡潔に後述される。

【０１０３】わずかな考えによって、無視されるならば、この実施例では、バックアップ動作がいくつかの複写ファイルを識別することができないようにし、したがって記憶要求にわずかに影響を及ぼすようになる小さい問題があることが明かになる。二人のユーザが同時にバックアップを実行するか、（又は実際に一方のユーザの

54

バックアップファイルがエージェント108によって＼BACKUP＼USERパス121から＼BACKUP＼SYSTEMパス122に移動される前に他方のユーザが開始するならば、）どちらのユーザも、他方から複写ファイルを識別することができない。これは、問題は決して完全にはなくならないけれども、最初の少数のユーザが自分の初期のバックアップを実行しているときに発生する“初期設定”期間中のたぶん最大の問題である。好ましい実施例ではこの問題のためのワークアラウンドは、エージェント108を移動処理の一部としてのある種の付加的複写ファイル“消去”を実行させることにある。これは、バックアップディレクトリファイルの内容を修正しないで行うことができる。その代わりに、複写ファイルのための<fileInfo>エントリは、全ファイルを包含している単一の<externPtr>参照420を含むように変更される。パフォーマンスの理由により、ネットワークがより少ないトラフィックを有するとき、この操作は、実際は、真夜中のようなより遅い時間まで延期してもよい。この問題が、特に、初期のグローバルデータベースを形成するために少数の代表的なノードにその初期のバックアップを逐次実行させることによってインストール後にアドミニストレータが“援助する”かどうかについて心配するほど単に重要ではないことは実際に有り得ることである。したがって、好ましい実施例では、アドミニストレータはこの機能性を使いものにならないようにすることができる。

【０１０４】ある場合には、ユーザは、所定のバックアップセットを削除することを望み、一般的にはバックアップ記憶手段101上の領域を節約することができる。例えば、ユーザは、２、３月経った後古い毎日のバックアップを毎週（又は毎月）バックアップに合併しようと決心してもよい。好ましい実施例の複写ファイルID及びファイル差分のために、得られるディスク節約は通常かなり小さい。好ましい実施例では、バックアップアプリケーションは、エージェント108を要求するファイルに通知し、削除を実行する。この削除は、残りのバックアップセットによってこのユーザ又は他のユーザのいずれかが参照されるいかなるファイル及びディレクトリエントリのコピーも保持するために、いくつかのバックアップセットを単一のバックアップディレクトリ／データファイルセットに統合することを含んでもよい。この統合操作は、ネットワーク上で非ビジー時まで延期することが最もよい。統合操作の完了は、いかなるユーザも当のファイルへの参照を含むバックアップセットをマウントしなくなるまで延期しなければならないこともある。ファイル参照及びディレクトリ参照の両方のためのインデックス（直接ポインタの代わりに）の使用はこのような操作を非常に簡単にすることを確認せよ。このような統合は、この余分なレベルの間接的な処置なしに依然として実行することができるが、一般に、この実施例で生じる単一の“スタブ”バックアップファイルセットの作成の

55

代わりに、残りのバックアップファイルの多くのための時間のかかる変更を含む。

【０１０５】５．アドミニストレータ機能

ネットワークアドミニストレータであってもよいし、なくてもよいバックアップドミニストレータは、好ましい実施例では実行するのにいくつかの機能を有する。これらの機能は、インストール時に費やされたたいていの努力によって大部分明白であるべきであるが、時々、他の決定及び動作が必要になることもある。

【０１０６】５．１　インストール

バックアップアドミニストレータは、エージェントコンピュータ107（自分自身のデスクトップであってもよい）にバックアップソフトウェアをインストールすべきであり、バックアップファイルが記憶すべきであるネットワークディレクトリ構造（例えば、＼BACKUP＼SYSTEM 122及び＼BACKUP＼USER 121）を設定すべきである。初期のディレクトリ構造及びアクセス権の設定は、バックアップアドミニストレータのネットワークアクセス権に応じて、ネットワークアドミニストレータからのある種のヘルプを伴ってもよい。

【０１０７】バックアップソフトウェアは、CD－ROM又はフロッピーディスクのいずれかで提供されるが、好ましい実施例のソフトウェアは、このようにネットワーク上でインストールされ、ユーザがネットワークからのセットアッププログラムを実行するようにするので、一般にバックアップアドミニストレータだけは常に提供媒体を使用しなければならない。できるだけ広範囲にわたって、インストールは自動化される。すなわち、アドミニストレータは、＼BACKUPディレクトリ120が存在し、ソフトウェアがインストールされているインストールソフトウェアだけに知らせなければならない。

【０１０８】５．２　新しいユーザの追加

インストールされたバックアップソフトウェアによって、ユーザが実際にいかなるバックアップを実行する前にも、アドミニストレータはUserAccountDatabase146内にユーザのための“アカウント”をセットアップしなければならない。これは２つの理由のために重要である。第一に、各ユーザは、自分自身のディレクトリ（例えば、125、129）及びユーザにわたって共有されるファイルを識別する際に重要であるユニークな<user Index>番号を有しなければない。第二に、アカウントデータベースを保持することによって、アドミニストレータは、システムへのアクセスを制限し、ライセンスの期間によりソフトウェアの使用量を計測するようにする。

【０１０９】好ましい実施例では、新しいユーザアカウントを追加することの一部として、アドミニストレータは、適当なアクセス権（また一方、これはネットワークアドミニストレータ権を必要とする）に関するユーザディレクトリを作成する。各ユーザは、ジョン（JOHN）のようなアドミニストレータによって選択されたユニーク

56

なユーザ名及びユニークな16ビット<user Index>（ユーザが決して直接知る必要がない）を割り当てられる。

好ましい実施例ではユーザ名の代わりに<user Index>に基づいているユニークなユーザディレクトリ名（例えば、USER2）とともにこの情報は、全てのユーザに対して読み出し専用であるUserAccountDatabase146に書き込まれる。このUserAccountDatabaseの存在によって、ハッカーがいかなるユーザの<user Index>及びディレクトリ名も限定するようにするけれども（ある種のリバースエンジニアリングによって、これらの２つのフィールドは、暗号化されていない間は、好ましい実施例ではクリア内に記憶されていないので）、ユーザのパスワードが危険にさらされていないと仮定すると、このような情報は、多分バックアップセッションの頻度及びサイズの情報以外いかなる方法でもユーザのデータの機密を弱めないことに注目せよ。アドミニストレータは、userDirKey値及びuserPostKey値のようなユーザアカウントの専用フィールドを暗号化するために使用される初期パスワードをユーザにも割り当てる。好ましい実施例では、ユーザアカウントエントリは、アドミニストレータのパスワードで暗号化されたUserAccountDatabase146で複製されるので、ユーザが自分のパスワードを失念したとしても、キーは消失されない。

【０１１０】次に、アドミニストレータは、通常ユーザに電子メールによって、ユーザのアカウントが現在アクティブであることを知らせ、ユーザに割り当てたユーザ名及び（一時的な）パスワードを与える。次に、ユーザは、ユーザが簡単にアイコンをダブルクリックできるように電子メールメッセージに.EXEファイルを付加することによってマイクロソフトのウィンドウズ3.1の下で実行することができる＼BACKUP＼SYSTEM＼GLOBALディレクトリからのセットアッププログラムを実行する。ユーザは自分のアカウント及びパスワードを入力し、ソフトウェアは、パーソナルバックアップディレクトリを（一般的にユーザのローカルハードディスク上に）セットアップし、いかなる必要なファイルもそのディレクトリにコピーする。このパーソナルディレクトリは、ネットワーク帯域幅消費を最少にするために、前のディレクトリファイル（例えば、141）のような所定のファイルをキャッシュするためにも使用される。ユーザがそのように選択するならば、最少のプログラムファイルセットだけをコピーすることは可能であるので（たぶん望ましい）、ユーザはネットワークからソフトウェアの最新コピーを常に実行する。交互に、ソフトウェアは、最新であり、新しいバージョンが検出されたときユーザにアップグレードするための許可を求めることを確認するためにネットワーク上のバージョンとそのバージョンを照合する。

【０１１１】ユーザは、初期インストール中、自分のパスワードを変更することもできる。セットアップ手順中、ユーザは、どれくらいしばしば周期的なバックアッ

57

プを予定しているか及びパーソナルバックアップディレクトリはどこに存在すべきかのようないかなる関連の個人の好みをも入力するように照会される。ユーザ名及び<user Index>とともにこれらの好みはユーザ好みファイル（例えば、142）に記憶されている。たいていの好みは後で変更してもよい。

【０１１２】５．３　エージェントの監視

一般にエージェントタスクはいかなる監視もなしにバックグラウンドで実行する。しかしながら、エージェント処理108を再起動するためにアドミニストレータによってある種の介入を必要とすることができる（システムダウンのような）状況が生じる。一般にエージェント108が生じる大部分の問題を復旧することを意図しているが、完全な復旧可能性を保証することはたぶんできない。好ましい実施例のエージェント処理108は、アドミニストレータが見直しすることができるその活動のログファイルを生成する。アドミニストレータは、エージェント108が、タイミング良くそのタスクを実行し、疑わしく見えるいかなる活動（又はそれの欠如）をも観測することによって警告を与えていることを確認するためにいくつかの簡単なチェックを実行できる監視アプリケーションをも有している。

【０１１３】本発明は、典型的で、好ましい実施例で記載されているが、それに限定されるものではない。当業者は、多数の付加修正及び改良が本質的な精神及び範囲から逸脱しないで本発明に行われていることを認識している。本発明の範囲は付加されたクレームセットによってのみ限定されるべきである。

【図面の簡単な説明】

本発明の好ましい実施例は、同じ参照番号は同じ部品を示す下記の図面の中及び図面によって示されている。

【図１】本発明のバックアップシステムのための典型的なネットワーク構成を示すブロック図

58

【図２】バックアップファイルが記憶されている典型的なディレクトリ構造を示す図

【図３】本発明のバックアップディレクトリに含まれたファイルの種類を示すブロック図

【図４】本発明によるバックアップディレクトリファイル内のディレクトリエントリのフォーマットのバッカス-ナウア記法（BNF）記述

【図５】図４に規定されたディレクトリエントリフォーマットの特定例のアセンブリ言語記述

【図６】本発明によるバックアップデータファイルの配置のブロック図

【図７】本発明によるバックアップデータファイルのフォーマットのBNF記述

【図８】本発明によるバックアップデータファイル内の<seekPts>レコードの例を示すブロック図

【図９】本発明によるグローバルディレクトリデータベースファイルの配置を示す図

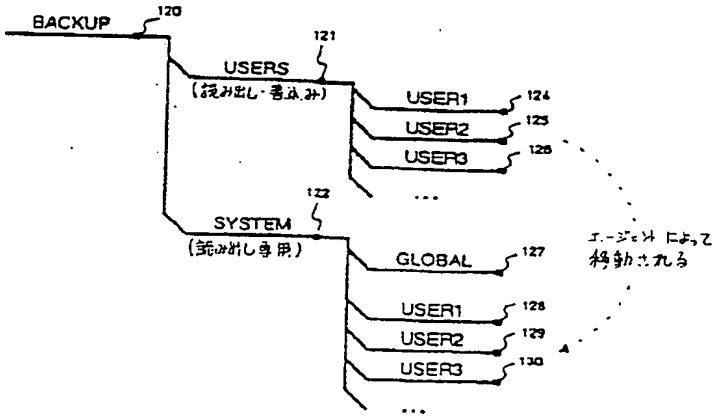
【図１０】本発明による第１のレベルのグローバルディレクトリデータベースを探索する際に使用されるデータ構造を示す図

【図１１】いかにユーザパスワードが本発明によるバックアップデータをアクセスするように暗号化キーをアクセスするために使用されるかを示す図

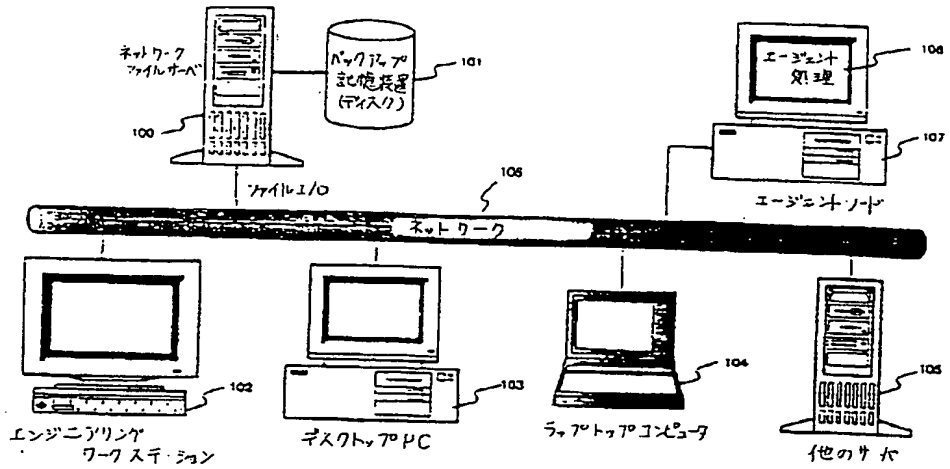
【符号の説明】

100　ネットワークファイルサーバ
101　バックアップ記憶装置
102　エンジニアリングワークステーション
103　デスクトップPC
104　ラップトップコンピュータ
105　他のサーバ
106　ネットワーク
107　エージェントノード
108　エージェント処理

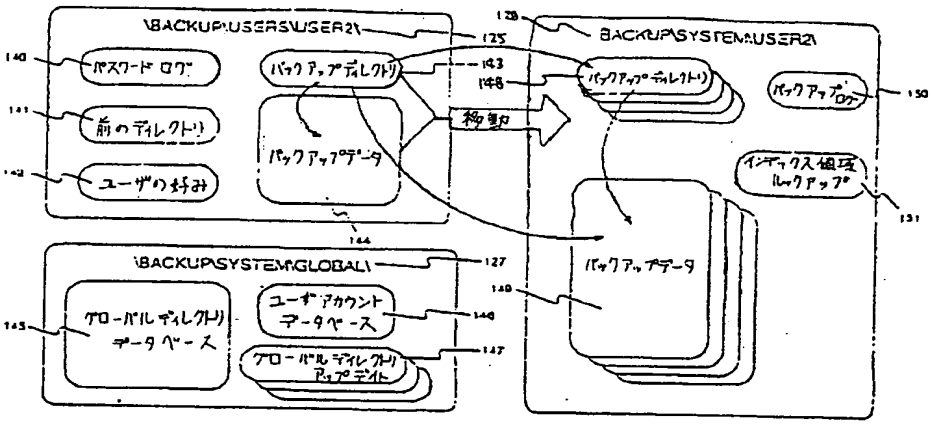
【図２】



【図１】



【図３】



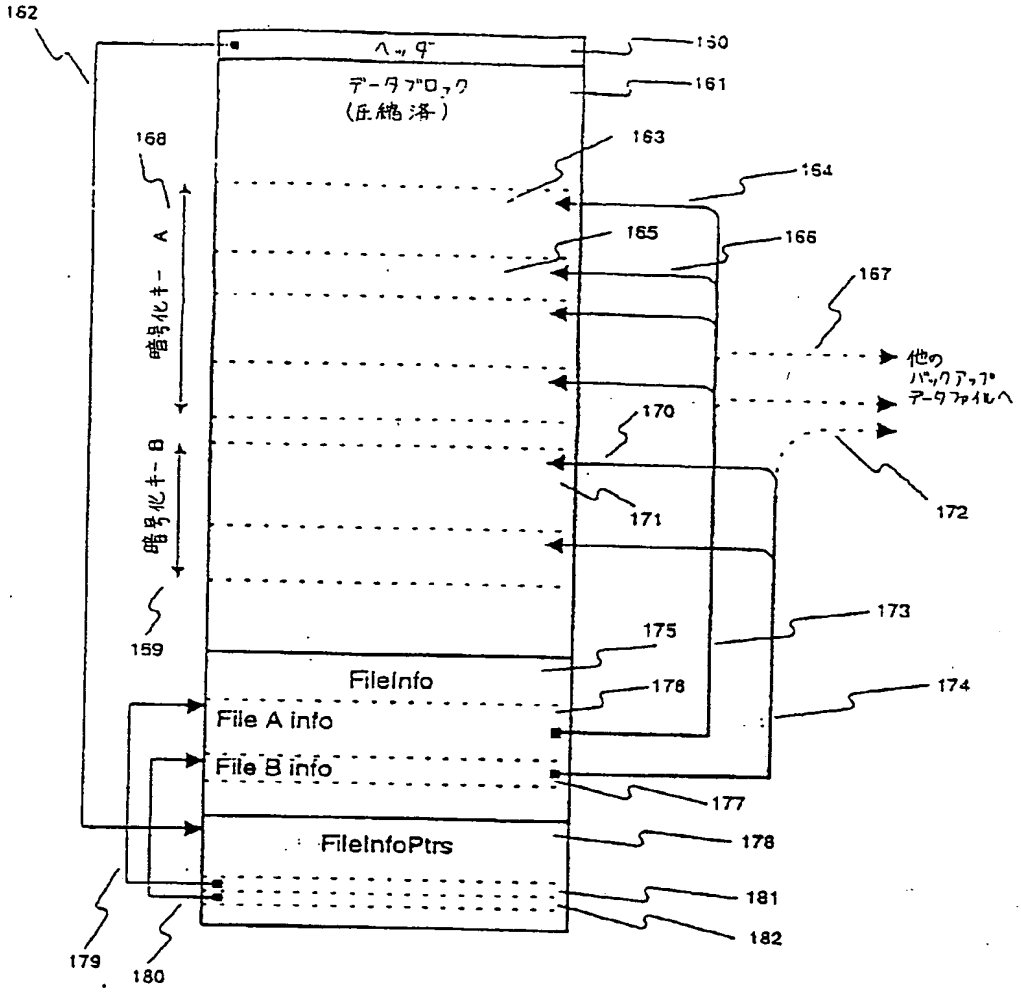
【図４】

```
200.    <volumeDirInfo>   ::= <subdirFileList>* [<externDirItem>]*
201.    <subdirFileList>  ::= [<fileEntry> | <subdirEntry>]* <endOfList> <externCount>
202.    <endOfList>       ::= 0x00                          // zero byte marks end of list
203.    <externCount>     ::= <itemCount>                   // references <externDirItems>
204.    <itemCount>       ::= <itemCount8> | <itemCount16>
205.    <itemCount8>      ::= 0x00 .. 0xFE                  //  8-bits: 0..254
206.    <itemCount16>     ::= 0xFF <word>                   // 24-bits: 0..65535
207.    <fileEntry>       ::= <fileName> <fileAttrib> <fileTime> <fileSize> <fileID>
208.    <subdirEntry>     ::= <dirName>  <fileAttrib> <fileTime>
209.    <fileAttrib>      ::= 0x00 .. 0xFF                  // attribute bits      for file/dir
210.    <fileTime>        ::= <dword>                       // creation time/date for file/dir
211.    <fileSize>        ::= <dword>                       // size of file in bytes
212.    <fileName>        ::= <ascii>                       // name of the file
213.    <dirName>         ::= <ascii>                       // name of the subdirectory
214.    <fileID>          ::= <fileIndex> <userIndex>       // which file, which user
215.    <fileIndex>       ::= <dword>                       // up to  4G files per user
216.    <userIndex>       ::= <word>                        // up to 64K users per system
217.    <word>            ::= 0x0000 .. 0xFFFF              // 16-bit unsigned quantity
218.    <dword>           ::= 0x00000000 .. 0xFFFFFFFF      // 32-bit unsigned quantity
219.    <ascii>           ::= [0x01 .. 0xFF]* 0x00          // zero-terminated char string
220.    <externDirItem>   ::= <oneItem> | <manyItems>       // unchanged dir entries
221.    <oneItem>         ::= 0 <dirItemNum>                // total of 32 bits
222.    <manyItems>       ::= 1 <dirItemNum> <itemCount>    // total of 32 bits + itemCount
223.    <dirItemNum>      ::= 0x80000001 .. 0xFFFFFFFF      // dir item number: 31 bits
```

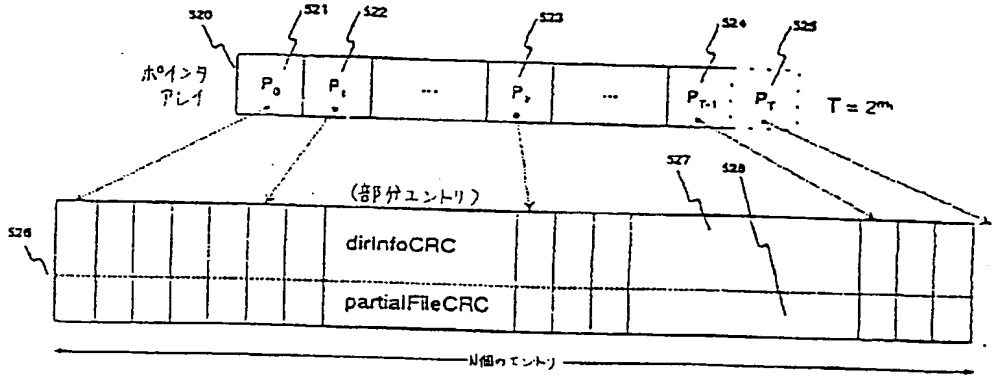
【図５】

```
FA_DIREC        =       10h                     ; bit 5 of attribute byte ==> directory

;<volumeDirInfo>: a small example illustrating the format of Figure 4
        ; <subdirFileList> for root directory:
        db      'DIR1',0                        ; <subdirEntry>:<dirName>: \DIR1
        db      FA_DIREC                        ;               <fileAttrib> for subdir
        dd      (?)                             ;               <fileTime>   for subdir
        db      'DIR2',0                        ; <subdirEntry>:<dirName>: \DIR2
        db      FA_DIREC                        ;               <fileAttrib> for subdir
        dd      (?)                             ;               <fileTime>   for subdir
        db      'FILE1',0                       ; <fileEntry>:  <fileName>:\FILE1
        db      0                               ;               <fileAttrib> for file
        dd      (?)                             ;               <fileTime>   for file
        dd      25000                           ;               <fileSize>
        dd      1187                            ;               <fileID>:<fileIndex>=1187
        dw      17                              ;                       <userIndex>=17
        db      0                               ; <endOfList>
        db      1                               ; <externCount> = 1 (item 29)

        ; <subdirFileList> for \DIR2:
        db      'SUBDIR_A',0                    ; <subdirEntry>:<dirName>: \DIR2\SUBDIR_A
        db      FA_DIREC                        ;               <fileAttrib> for subdir
        dd      (?)                             ;               <fileTime>   for subdir
        db      'FILE2',0                       ; <fileEntry>:  <fileName>:\DIR2\FILE2
        db      0                               ;               <fileAttrib> for file
        dd      (?)                             ;               <fileTime>   for file
        dd      10000                           ;               <fileSize>
        dd      1395                            ;               <fileID>:<fileIndex>=1395
        dw      17                              ;                       <userIndex>=17
        db      0                               ; <endOfList>
        db      2                               ; <externCount> = 2 (items 73,75)

        ; <subdirFileList> for \DIR2\SUBDIR_A:
        db      'FILE4',0                       ; <fileEntry>:  <fileName>:\DIR2\FILE3
        db      0                               ;               <fileAttrib> for file
        dd      (?)                             ;               <fileTime>   for file
        dd      1                               ;               <fileSize>
        dd      233                             ;               <fileID>:<fileIndex>= 233
        dw      1                               ;                       <userIndex>= 1
        db      0                               ; <endOfList>
        db      0                               ; <externCount> = 0 (no items)

        ; <subdirFileList> for \DIR1:
        db      'SUBDIR_B',0                    ; <subdirEntry>:<dirName>: \DIR1\SUBDIR_B
        db      FA_DIREC                        ;               <fileAttrib> for subdir
        dd      (?)                             ;               <fileTime>   for subdir
        db      'FILE3',0                       ; <fileEntry>:  <fileName>:\DIR1\FILE3
        db      0                               ;               <fileAttrib> for file
        dd      (?)                             ;               <fileTime>   for file
        dd      0                               ;               <fileSize> = 0 (no data)
        dd      0                               ;               <fileID>:<fileIndex>=NULL
        dw      0                               ;                       <userIndex>=NULL
        db      0                               ; <endOfList>
        db      3                               ; <externCount> = 3 (items 81,83,87)

        ; <subdirFileList> for \DIR1\SUBDIR_B:
        db      0                               ; <endOfList>: no explicit entries here
        db      2                               ; <externCount> = 2 (items 91,93-95)
        ; end of all <subdirFileList> records

        ;<externOtherItems>:
        dd      29                              ; <oneItem> : root dir
        dd      73,75                           ; <oneItem>*: \DIR2
        dd      81,83,87                        ; <oneItem>*: \DIR1
        dd      91                              ; <oneItem> : \DIR2\SUBDIR_B
        dd      93-80000000H                    ;   <manyItems>: <baseItemNum>=93
        db      3                               ;                 <itemCnt>=3  (93-95)
```

【図６】


162
ヘッダ
160
データブロック
(圧縮済)
161
168
163
164
165
166
167
暗号化キー A
暗号化キー B
他の
バックアップ
データファイルへ
170
171
172
159
175
173
FileInfo
176
174
File A info
File B info
177
FileInfoPtrs
178
181
182
179
180

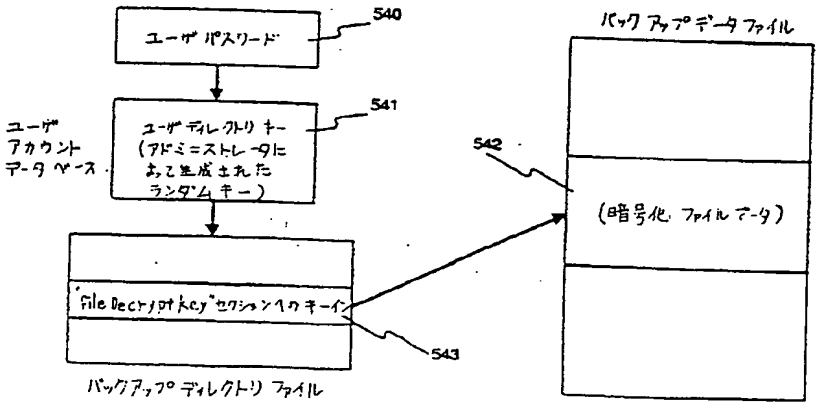
【図１０】


520
521
522
523
524
525
ポインタ
アレイ
P_0
P_1
...
P_x
...
P_{T-1}
P_T
$T = 2^m$
527
528
(部分エントリ)
526
dirInfoCRC
partialFileCRC
N個のエントリ

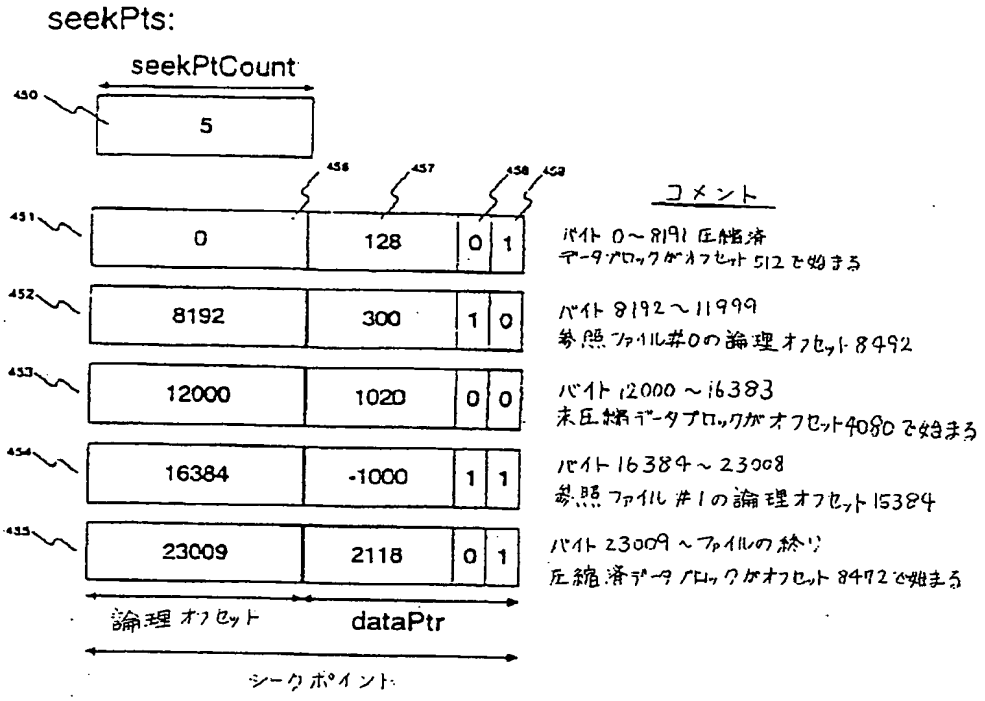
【図７】

```
400.  <bkupDataFile>   ::= <header> <dataBlock>* <fileInfo>* <fInfoPtrs>
401.  <header>         ::= ... <fInfoPtrOffset> <indexRangeCnt> ...
402.  <fInfoPtrOffset> ::= <dword>                    // where to find fInfoPtrs in file
403.  <indexRangeCnt>  ::= <dword>                    // # indexRange entries
404.
405.  <dataBlock>      ::= <byte>*                    // starts dword aligned in file
406.  <byte>           ::= 0x00 .. 0xFF               // 8-bit bytes
407.
408.  <fileInfo>       ::= <fileCRC> <bitFields> <seekPts> [<fileRef>*] <fprints>
409.  <fileCRC>        ::= <dword>                    // CRC over entire file contents
410.  <bitFields>      ::= <refCnt> <refLevel> <isGlobal>
411.  <refCnt>         ::= 00 | 01 | 10               // two bits: # referenced files
412.  <refLevel>       ::= 0x00 .. 0x3F               // max # ref levels of indirection
413.  <isGlobal>       ::= 0  | 1                     // boolean: new/updated file?
414.  <seekPts>        ::= <seekPtCount> <seekPoint>* // how to find data blocks of file
415.  <seekPtCount>    ::= <dword>                    // # seekpoints in fileInfo
416.  <seekPoint>      ::= <logicalOffset> <dataPtr>  // binary search on logicalOffset
417.  <logicalOffset>  ::= <dword>                    // logical file offset of block
418.  <dataPtr>        ::= <dataBlockPtr> | <externPtr>// internal/external ptr (32-bits)
419.  <dataBlockPtr>   ::= <blockOffs> 0 <packFlag>   // reference internal data block
420.  <externPtr>      ::= <relOffs>    1 <refFileNo>  // reference external file
421.  <blockOffs>      ::= <bits30>                   // offset of dataBlock in dwords
422.  <packFlag>       ::= 0 | 1                      // boolean: raw/compressed block?
423.  <relOffs>        ::= <bits30>                   // signed relative logical offset
424.  <refFileNo>      ::= 0 | 1                      // which reference file?
425.  <bits30>         ::= 0x00000000 .. 0x3FFFFFFF   // 30 bit field (part of a dword)
426.  <fileRef>        ::= <fileID> <decryptKey>      // which file, how to decrypt it
427.  <decryptKey>     ::= <dword> <dword>            // 64-bit private encryption key
428.  <fprints>        ::= <fpChunkSize> <fingerPrint>*// for chunk matching
429.  <fpChunkSize>    ::= <word>                     // # bytes covered by fingerprint
430.  <fingerPrint>    ::= <dword> <dword> <dword>    // 96 bits as chunk signature
431.
432.  <fInfoPtrs>      ::= <indexRange>* <fileInfoData>*
433.  <indexRange>     ::= <indexBase> <indexCount>   // # indexRange entries in header
434.  <indexBase>      ::= <dword>                    // first file index in range
435.  <indexCount>     ::= <dword>                    // # indices in this range
436.  <fileInfoData>   ::= <fileInfoPtr> <fileSize> <dirInfoCRC> <partialFileCRC>
437.  <fileInfoPtr>    ::= <dword>                    // offset of <fileInfo> record
438.  <fileSize>       ::= <dword>                    // file size in bytes
439.  <dirInfoCRC>     ::= <dword>                    // CRC over file directory entry
440.  <partialFileCRC> ::= <dword>                    // CRC over first 256K of file
```

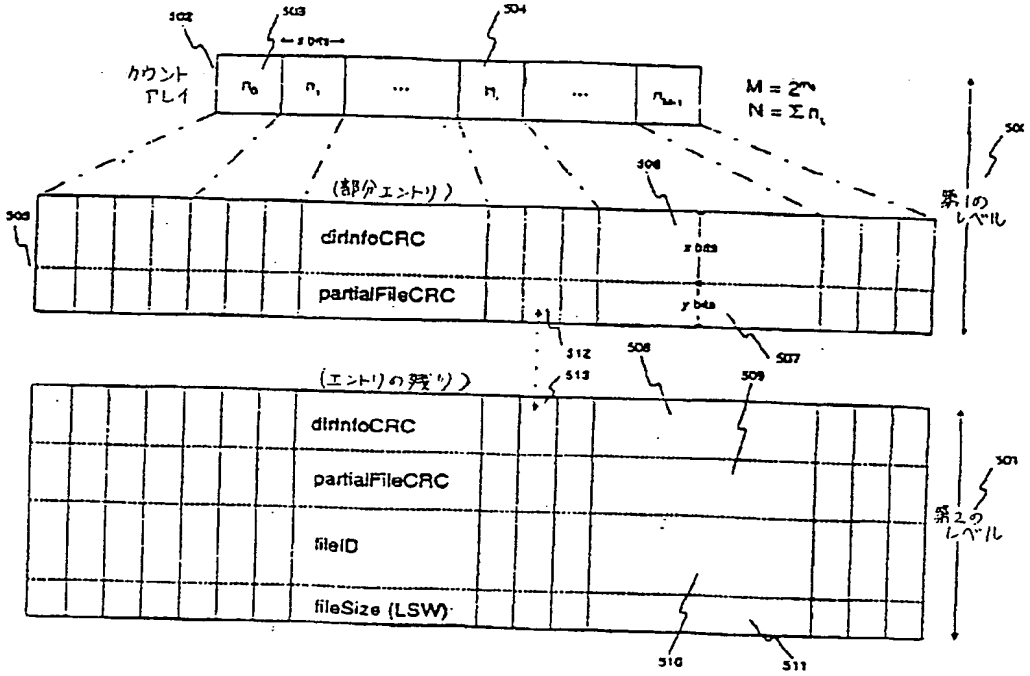
【図１１】



【図８】


seekPts:
seekPtCount
5
456
457
458
459
451
452
453
454
455
0
128
0
1
8192
300
1
0
12000
1020
0
0
16384
-1000
1
1
23009
2118
0
1
論理オフセット
dataPtr
シークポイント
コメント
バイト 0～8191 圧縮済
データブロックがオフセット 512 で始まる
バイト 8192～11999
参照ファイル#0の論理オフセット 8492
バイト 12000～16383
未圧縮データブロックがオフセット4080で始まる
バイト 16384～23008
参照ファイル #1の論理オフセット 15384
バイト 23009～ファイルの終り
圧縮済データブロックがオフセット 8492で始まる

【図９】


カウント
アレイ
M = 2^m
N = Σ n_i
(部分エントリ)
dirInfoCRC
partialFileCRC
(エントリの残り)
dirInfoCRC
partialFileCRC
fileID
fileSize (LSW)
第1のレベル
第2のレベル

フロントページの続き

(72)発明者　トム　ディラトゥッシュ
アメリカ合衆国　カリフォルニア州
91911　チュラ　ヴィスタ　カイアマック
アヴェニュー　1052